The Thesis Committee for Jeffery Shipeng Luo

Certifies that this is the approved version of the following thesis:

X-Ray Monitoring in Heterogeneous and Fractured Porous Media: Experimental Measurement and Numerical Modeling

APPROVED BY

SUPERVISING COMMITTEE:

David Nicolas Espinoza, Supervisor

David DiCarlo

X-Ray Monitoring in Heterogeneous and Fractured Porous Media: Experimental Measurement and Numerical Modeling

by

Jeffery Shipeng Luo

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2019

Acknowledgements

I would like to thank Dr. David Nicolas Espinoza, Dr. David DiCarlo, and Dr. Quoc Phoc Nguyen for their supervision and guidance throughout this research endeavor. I would also like to thank Dr. Xiongyu Chen for his expertise. Gary Miscoe and Daryl Nygaard were also very helpful in the helping me design my experimental apparatuses.

Abstract

X-Ray Monitoring in Heterogeneous and Fractured Porous Media: Experimental Measurement and Numerical Modeling

Jeffery Shipeng Luo, M.S.E.

The University of Texas at Austin, 2019

Supervisor: David Nicolas Espinoza

The complexity of fluid flow in heterogeneous and fractured media has long been a focus in the energy industry. Models have been developed in an attempt to determine the effects of rock heterogeneity, including fractures and laminations, on hydrocarbon flow. These models rely on an integration of data from different scales, including but not limited to seismic, core, and pore-level. This study focuses on the integration of studies at the core and pore-level through X-ray analysis of three unique projects.

The first project addresses gas mobility control through X-ray micro-focus visualization of WAG core-flood experiments and interpretation aided by numerical simulation. We use surfactant as our primary mobility control agent to stabilize the nitrogen gas dispersion during WAG injection. We quantify the improvement in sweep efficiency by utilizing an automated fluid injection system monitored by an X-ray micro-focus scanner to quantify how displacement patterns and water saturation change with time. The

core-flood device is wirelessly operated through a computer. The resolution of the images permits observation of not only core scale fingering but also pore-scale displacement. Results show that saturation patterns and displacement front during WAG injection are highly influenced by bedding orientation and rock heterogeneity. Without gas mobility control during WAG injection, fingering and early breakthrough occur in those cases in which bedding orientation facilitates gas to flow through high permeability layers. In these cases, sweep efficiency is low during early time injection of nitrogen and only improves after injection is prolonged. With gas mobility control, the displacement efficiency is significantly improved. Simulation work matches experimental data well and replicates saturation patterns measured experimentally in laminated Berea sandstone samples.

The second project focuses on understanding how pre-existing fractures and rock heterogeneities can aid the propagation of induced fractures by 3D mapping of fracture networks in Mancos shale core plugs using X-ray micro-CT. Analysis of both intact samples and samples with pre-existing fractures show that induced fractures tend to develop along the same orientation of lamination planes, which more often than not correspond to the same orientation of any pre-existing fractures. The nature of this work requires inspection of fine fracture networks within larger specimens, so scanning at these coarser resolutions to capture the sample in their entirety leads to a compromise on the range of fracture sizes that can be accurately visualized. Moreover, existing limitations of the technique, including blurring effects further complicate interpretation. This project reviews some of these issues and remedies to overcome these limitations.

The third project uses X-ray micro-CT to monitor methane hydrate growth and dissociation experiments in sands partially saturated with KI brine under excess-gas and excess-water conditions. The experiments show coexistence of brine, gas, and hydrate at the pore-scale and their evolution towards three-phase equilibrium. Analysis reveals that hydrate first forms as a porous hydrate mixing of brine and gas and then evolves into separate phases as hydrate cages exclude ions. During this time, hydrate growth mobilizes water over long distances. This mobilization of water is critical to explaining heterogeneous hydrate distribution as a result of hydrate ripening.

The novel visualization of various pore-scale phenomenon presented here provides new pore-scale experimental insight to the structure and flow behavior of various heterogeneous mediums at a resolution one order of magnitude higher than with medical X-ray CT or other core-scale visualization techniques. The findings are useful for understanding complex flow patterns and structures of heterogeneous mediums.

Table of Contents

Acknowledgements iv
Abstract v
Table of Contents viii
List of Tables xi
List of Figures xii

Chapter 1: X-Ray Micro-Focus Monitoring of Water Alternating Gas Injection in Heterogeneous Berea Sandstone Samples 1
1.1 Introduction 1
1.2 Materials and Methods 3
1.2.1 Rock and Fluid Types 3
1.2.2 Flow Automation and Data Logging 4
1.2.2a Experimental Setup 4
1.2.2b Electronics for Automated System 8
1) Arduino Uno R3 8
2) HC-06 Bluetooth Module 9
3) Jbtek 4 Channel Relay Module 11
4) 12V DC Solenoid Valves 13
5) Autex Pressure Transducer 15
6) Adafruit Micro-SD Breakout Board (ADA254) 16
7) DS 3231 Real Time Clock Module 17
1.2.2c Arduino IDE for Automated System 18
1.2.2d Android App and Tera Term for Bluetooth Communication 20
1) Android App 20
2) Tera Term 24
1.2.3 X-Ray System 25
1.2.4 Image Processing and Analysis 26
1.2.5 Permeability Measurement 28
1.2.6a Unsteady-State Permeability Measurement 28
1.2.6b Steady-State Permeability Measurement 33
1.2.6c Layer Permeability Calculation 33
1.3 Image Optimization and Calibration 34
1.3.1 Image Optimization 34
1.3.1 Calibration Techniques 36
1.3.1a Continuous Flux Normalization 36
1.3.1b Initial Flux Normalization 40
1.4 Results and Discussion 42
1.4.1 WAG Injection Parallel to Bedding Direction 42
1.4.2 WAG Injection Perpendicular to Bedding Direction 42
1.4.3 Effect of Surfactant-Stabilized Foam 45
1.5 Numerical Simulations 51
1.5.1 Experimental Model Description 51

1.5.2 Fluid Component Data 51
1.5.3 Rock Fluid Data 52
1.5.4 Well Data 55
1.5.5 Results 56
1.5.5a Vertical Bedding Direction 56
1.5.5b Horizontal Bedding Direction 57
1.6 Conclusions 60

Chapter 2: 3D Mapping of Fracture Networks in Shale Cores 61
2.1 Introduction 61
2.2 Materials and Methods 62
2.2.1 Rock Types 62
2.2.2 Triaxial Load Frame and Testing Conditions 63
2.2.3 X-Ray System 64
2.2.4 Image Processing and Analysis 65
2.3 Results 75
2.3.1 Perpendicular Bedding at 0° 75
2.3.2 Direction Bedding at 45° 76
2.3.3 Parallel Bedding at 90° 77
2.4 Discussion 78
2.4.1 Effect of Bedding Orientation and Pre-Existing Fractures 78
2.4.2 Effect of Bedding Orientation and Confining Stress 81
2.4.3 Limitations in Identifying Fractures through Image Segmentation 82
2.5 Conclusions 86

Chapter 3: X-Ray Micro-CT Observation of Methane Hydrate Growth and Dissociation in Sandy Sediments 87
3.1 Introduction 87
3.2 Materials and Methods 90
3.2.1 Rock and Fluid Types 90
3.2.2 PID Thermoelectric Cooling and Data Logging 90
3.2.2a Experimental Setup 90
3.2.2b Hydrate Formation 96
3.2.2c Initial Gas Loading Pressure 97
3.2.2d Electronics for Automated System 98
1) Elegoo Mega 2560 R3 98
2) Adafruit Micro-SD Breakout Board (ADA254) 99
3) DS 3231 Real Time Clock Module 99
4) Peltier Cell 100
5) 12V Brushless DC Cooling Fan 101
5) MDD10A Motor Driver 102
6) LCD 1602 Module 103
7) Thermistor 103
3.2.2e Arduino IDE for PID Thermoelectric Cooling 104

3.2.3 X-Ray System 106
3.2.4 Image Processing and Analysis 106
3.3 Results and Discussion 108
3.3.1 Overview 108
3.3.2 Water-to-Hydrate Conversion Ratio 111
3.3.3 Three-Phase Brine-Gas-Hydrate 112
3.3.4 Gas and Water Mobility During Hydrate Growth 113
3.3.5 Hydrate Pore-Habit and Micro-Morphology 117
3.3.6 Gas Hydrate Dissociation 118
3.4 Conclusions 121
Appendices 122
Appendix A: Arduino IDE Sketches 122
A.1 Project 1: Flow Automation and Data Logging 122
A.2 Project 1: Syringe Pump 127
A.3 Project 3: PID Thermoelectric Cooling and Data Logging 130
Appendix B: MATLAB Code 141
B.1 Project 1: Water Saturation Map for WAG Analysis 141
B.2 Project 1: Average Water Saturation for WAG Analysis 145
B.2 Project 3: Initial Gas Loading Pressure for Excess-Water Hydrate Experiment 150
Appendix C: Syringe Pump 166
C.1 Hardware for Syringe Pump 167
1) NEMA Stepper Motor 167
2) Electronics 169
3) Motor Mount 169
4) Shaft Coupler 169
5) Syringe Plunger Mount 170
6) Syringe Plunger 171
7) Syringe Barrel Holder 171
8) Syringe Barrel 171
9) Syringe Tip Holder 171
10) Mounting Rail 172
11) M8 Threaded Rod 172
12) Smooth Rod 172
C.2 Electronics for Syringe Pump 173
1) Arduino Uno R3 174
3) HC-06 Bluetooth Module 175
3) Big Easy Driver 176
4) Proto-Screwshield (Wingshield) 177
C.3 Arduino IDE for Syringe Pump 177
C.4 Android App for Bluetooth Communication 179
References 184

List of Tables

Table 1.1—Pressure transducer error from calibration.....16
Table 1.2—Permeability values from laboratory measurements.....28
Table 1.3—Properties of nitrogen gas. 52
Table 1.4— Values used in Honarpour et al. correlations. 53
Table 1.5—Water-oil and gas-liquid table for rock type 1 (coarse sandstone). 54
Table 1.6—Water-oil and gas-liquid table for rock type 2 (tight sandstone). 54
Table 2.1—Fracture network characterization summary table. 78
Table 3.1—Summary of initial measurements for hydrate experiments 108
Table 3.2—Summary of final measurements for hydrate experiments 108

List of Figures

Fig. 1.1—Improved sweep efficiency, reduced effects of gravity segregation and heterogeneities in FAWAG. 1
Fig. 1.2—Berea sandstone samples. 3
Fig. 1.3—Fluid injection schematic setup. 4
Fig. 1.4—X-ray radiography of epoxied 2D sandstone slab. 5
Fig. 1.5—Circuit schematic of flow automation and data logging system. 6
Fig. 1.6—Fritzing diagram of flow automation and data logging system. 7
Fig. 1.7—Fritzing breadboard (left) and circuit (right) schematic of Arduino Uno. 8
Fig. 1.8—Voltage divider schematic. 10
Fig. 1.9—Fritzing breadboard (top) and circuit (bottom) schematic of HC-06 bluetooth module. 10
Fig. 1.10—Picture of JBtek 4 Channel DC 5V Relay Module. 11
Fig. 1.11—Circuit schematic of JBtek Relay Module with jumper intact. 12
Fig. 1.12—Circuit schematic of JBtek Relay Module with jumper removed. 13
Fig. 1.13—Fritzing breadboard (top) and circuit (bottom) schematic of solenoid valve. 13
Fig. 1.14—Circuit schematic of solenoid valve setup. 14
Fig. 1.15— Fritzing breadboard (top) and circuit (bottom) schematic of diode. 14
Fig. 1.16—Pictures (left and middle) and Fritzing breadboard (right) schematic of pressure transducer. 15
Fig. 1.17—Pressure transducer calibration. 15
Fig. 1.18—Fritzing breadboard (left) and circuit (right) schematic of micro-SD board. 17
Fig. 1.19— Fritzing breadboard (left) and circuit (right) schematic of real time clock. 17
Fig. 1.20—Screenshot of ‘WAG’ Android application with markers A-H. 20
Fig. 1.21—Logic blocks corresponding to ‘Bluetooth_List’ or marker ‘A’. 21
Fig. 1.22—Logic blocks corresponding to ‘Clock’ and ‘Connected_Label’ or marker ‘B’. 22
Fig. 1.23—Screenshot of ‘WAG’ Android application indicating the Bluetooth is activated. 22
Fig. 1.24—Logic blocks corresponding to markers C-E. 23
Fig. 1.25—Logic blocks corresponding to markers F-H. 23
Fig. 1.26—Setting up connection between computer and Arduino via Bluetooth using Tera Term. 24
Fig. 1.27—Real-time monitoring of time-stamped pressure measurements and flow automation. 24
Fig. 1.28—Picture of epoxied sandstone slab and flow automation system inside X-ray micro CT machine. 25
Fig. 1.29—X-ray radiography image of horizontal bedding sandstone slab (left) and corresponding binary image (right). 28
Fig. 1.30—Unsteady state measurement setup from Jones (1972). 29

Fig. 1.31—Unsteady state pressure falloff during nitrogen injection for permeability and Klinkenberg effect correction for a vertical bedding sample. 30
Fig. 1.32—Steady state permeability calculation method. 33
Fig. 1.33—Oversaturated image (left) versus optimized image (right). 35
Fig. 1.34—Oversaturated image histogram (left) versus optimized image histogram (right). 35
Fig. 1.35—Calibration regions used to determine correctional factor used for manual calibration. 37
Fig. 1.36—Average grayscale number of calibration regions as a function of time before calibration (left) and after calibration (right). 37
Fig. 1.37—Water saturation measurements of H3 horizontal bedding before calibration (left) and after calibration (right) during primary drainage. 38
Fig. 1.38—Average grayscale number of left epoxy calibration regions as a function of time before calibration (left) and after calibration (right). 39
Fig. 1.39—Average grayscale number of rock as a function of time before and after calibration. 39
Fig. 1.40—Grayscale histogram of wet-dry image before calibration (left) and after calibration (right). 41
Fig. 1.41—Water saturation measurements of V2 vertical bedding (left) and H4 horizontal bedding (right) during primary drainage. 43
Fig. 1.42—Water saturation measurements of V2 vertical bedding (left) and H4 horizontal bedding (right) during primary imbibition. 44
Fig. 1.43—Average water saturation of V2 vertical bedding and H4 horizontal bedding during 4 WAG cycles (larger water saturation in vertical bedding case is due to a more even displacement front). 45
Fig. 1.44—Water saturation measurements of H4 horizontal bedding without surfactant (left) and H4 horizontal bedding with surfactant (right) during primary drainage. 46
Fig. 1.45—Water saturation measurements of H4 without surfactant (left) and H4 with surfactant (right) during primary imbibition. 47
Fig. 1.46—Water saturation measurements of H4 without surfactant (left) and H4 with surfactant (right) during secondary drainage. 49
Fig. 1.47—Average water saturation images of H4 without surfactant and H4 with surfactant during 4 WAG cycles. 50
Fig. 1.48—Water-oil relative permeability graph. 55
Fig. 1.49—Gas-liquid relative permeability graph. 55
Fig. 1.50—Water saturation simulation image during primary gas injection (from left to right). 56
Fig. 1.51—Water saturation images during primary gas injection (from left to right). 56
Fig. 1.52—Water saturation simulation image during primary gas injection (from left to right). 57
Fig. 1.53—Water saturation CT image during primary gas injection in type-2 rock (from left to right). 57

Fig. 1.54—Water saturation predictions of V2 vertical bedding (left) and H4 horizontal bedding (right) during primary drainage. 59
Fig. 2.1—Modified figure from Ramos (2018) showing Mancos core samples with bedding at 0° (left), 45° (middle), and 90° (right). 62
Fig. 2.2—Pictures of the Terratek triaxial load frame (left), triaxial cell (middle), and the corresponding schematic of the triaxial cell (right). 63
Fig. 2.3—Image analysis workflow using Fiji/Image-J to map fracture network. 65
Fig. 2.4—Stack of images are read (left), unnecessary slices removed (middle), and contrast adjusted (right). 66
Fig. 2.5—Original stack of images (left) and stack after contrast is adjusted and end slices removed. 66
Fig. 2.6—Segmentation results from two different threshold limits. The greater the limit the more noise. 68
Fig. 2.7—Processed stack of images (left) and stack after segmentation. 69
Fig. 2.8—Background is subtracted (left) and another threshold is applied (right). 69
Fig. 2.9—Segmented stack (top left), stack after subtracting background (top right), and stack after another threshold is applied to remove the outer ring (bottom right). 70
Fig. 2.10—Isolated fracture stack (left) and stack after inversion (right). 71
Fig. 2.11—3D viewer settings for fracture visualization (left) and its corresponding rendering (right). 72
Fig. 2.12—Superimposing segmented fracture onto background rock using Image Calculator. 73
Fig. 2.13—Isolated fracture stack (left) and stack after inversion (right). 73
Fig. 2.14—Merging segmented fracture with superimposed stack to further accentuate fracture network. 74
Fig. 2.15—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right). 75
Fig. 2.16—Fracture network shown in red superimposed on rock volume (left) and orthoslice (right). 75
Fig. 2.17—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right). 76
Fig. 2.18—Fracture network shown in red superimposed on rock volume (left) and orthoslice (right). 76
Fig. 2.19—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right). 77
Fig. 2.20—Fracture network shown in red (right) superimposed on rock volume (left)...77
Fig. 2.21—Fracture network shown in red (right) superimposed on rock volume (left)...79
Fig. 2.22—Parallel bedding sample (PL_6, left) and 45° bedding sample (45_1, right) (Ramos 2018). 80
Fig. 2.23—Parallel bedding sample (PL_7, left), perpendicular bedding sample (PD_15, middle), and 45° bedding sample (45_5, right) (Ramos 2018). 80
Fig. 2.24—Fracture network shown in red (right) superimposed on rock volume (left)...81

Fig. 2.25—Segmentation results from two different threshold limits. The greater the limit the more noise. 82
Fig. 2.26—Beam hardening correction results from three different presets.....84
Fig. 2.27—Original fracture network (left), with higher threshold limit (middle), and after cropping (right).....85
Fig. 3.1—Illustration of the three major types of pore habit models showing hydrate saturation of ~30%.....88
Fig. 3.2—Schematic of experimental setup with micro-consolidation cell and thermoelectric cooling capability.....91
Fig. 3.3—Picture of experimental setup, electrical ensemble out of frame and shown separately in Fig. 3.5.....91
Fig. 3.4—Picture of electrical components for experimental setup.....93
Fig. 3.5—Fritzing diagram of electrical ensemble for methane hydrate generation (Arduino Mega variant) 94
Fig. 3.6—Fritzing diagram of electrical ensemble for methane hydrate generation (Arduino Uno variant).....95
Fig. 3.7—Fritzing breadboard schematic (left) and picture (right) of Elegoo Mega 2560.....98
Fig. 3.8—Fritzing breadboard schematic (left) and picture (right) of micro-SD board. ...99
Fig. 3.9—Fritzing breadboard schematic (left) and picture (right) of real time clock. ...100
Fig. 3.10—Fritzing breadboard schematic (left) and picture (right) of Peltier cell. 100
Fig. 3.11—Thermoelectric cooling using Peltier cells stable at ~3 °C for more than 60 hours.....101
Fig. 3.12—Fritzing breadboard schematic (left) and picture (right) of brushless DC cooling fan. 101
Fig. 3.13—Fritzing breadboard schematic (left) and picture (right) of MDD10A dual motor driver 102
Fig. 3.14—Fritzing breadboard schematic (left) and picture (right) of LCD 103
Fig. 3.15—μCT images of sand saturated with methane gas and KI brine at known salinities (top left) and their respective histograms (bottom left). CT number after contrast adjustment versus salinity for KI brine and corresponding linear fit (right).....107
Fig. 3.16—Original (left) and segmented (right) μCT images of sand saturated with methane gas and 4.4 wt% KI brine (Chen et al. 2018a). Both are cropped μCT images of conditions outside of methane hydrate stability.....107
Fig. 3.17—Pressure-temperature path for excess-water experiments #1 and #2. Methane hydrate stability curves are given for 4.4 wt% and 30 wt% KI brine (Tishchenko et al. 2005). 109
Fig. 3.18—YZ orthoslices of μCT images taken for excess-water hydrate experiment #1.....109
Fig. 3.19—YZ orthoslices of μCT images taken for excess-water hydrate experiment #2.....110

Fig. 3.20—μCT and segmented images for excess-water KI experiment #2 (left) and excess-gas NaBr experiment (right) from Chen et al. (2018a). 112
Fig. 3.21—Axial CT slices of excess-gas experiment (left) from Chen et al. (2018a) and excess-water experiment #2 (right) showing conditions out of hydrate stability zone with sand partially saturated with brine/methane and conditions within stability zone with sand saturated with aggregated, interconnected hydrate. 114
Fig. 3.22—YZ orthoslice μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2. The images show pore-filling hydrate mixed with brine that evolve into separate porous hydrate and high salinity brine phases. Axial μCT and segmented slices of this experiment (dashed red line) shown in Fig. 3.23. 114
Fig. 3.23—Axial μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-gas experiment #2 (top) from Chen et al. (2018a) and excess-water experiment #2 (bottom). The images show pore-filling hydrate mixed with brine that evolve into separate porous hydrate and high salinity brine phases.....115
Fig. 3.24—Axial μCT slices of excess-gas experiment (top) from Chen et al. (2018a) showing more aggregated hydrate-rich phase within the bottom PTFE spacer, which did not contain water initially, than excess-water experiment #1 (bottom). As more hydrate grows, more high-salinity brine (white) appears within the hydrate aggregate and the porosity within hydrate becomes apparent (black). 116
Fig. 3.25—Short-lived brine menisci across grains during methane hydrate growth of excess-water experiment #1. 116
Fig. 3.26— Pressure-temperature path (left) and pressure decay curve for dissociation experiment. Dissociation started 18.2 days into excess-gas experiment #2.....118
Fig. 3.27—YZ orthoslice μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2 during dissociation. Axial μCT and segmented slices of this experiment (dashed red line) shown in Fig. 3.28. 119
Fig. 3.28—μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2 during dissociation. 120
Fig. C.1—Syringe pump with hardware components (1) NEMA motor (2) electronics (3) motor mount (4) shaft coupler (5) syringe plunger mount (6) syringe plunger (7) syringe barrel holder (8) syringe barrel (9) syringe tip holder (10) mounting rail (11) M8 threaded rod (12) smooth rod. 166
Fig. C.2—Syringe pump component: NEMA motor. 167
Fig. C.3—Syringe pump component: motor mount. 169

Fig. C.4—Syringe pump component: shaft coupler. 170
Fig. C.5—Syringe pump component: syringe plunger mount. 170
Fig. C.6—Syringe pump component: syringe barrel holder. 171
Fig. C.7—Syringe pump component: syringe tip holder. 172
Fig. C.8—Fritzing diagram of syringe pump electrical components. 173
Fig. C.9—Fritzing breadboard schematic of Arduino Uno. 174
Fig. C.10—Voltage divider schematic. 176
Fig. C.11—Fritzing breadboard schematic of HC-06 bluetooth module. 176
Fig. C.12—Picture of Big Easy Driver (left) and proto-screwshield (right). 177
Fig. C.13—Screenshot of 'SyringePump' Android application with markers A-F. 179
Fig. C.14—Logic blocks corresponding to 'Bluetooth_List' or marker 'A'. 180
Fig. C.15—Logic blocks corresponding to 'Clock' and 'Connected_Label' or marker 'B'. 181
Fig. C.16—Screenshot of 'WAG' Android application indicating the Bluetooth is activated. 181
Fig. C.17—Logic blocks corresponding to marker C. 182
Fig. C.18—Logic blocks corresponding to marker D. 182

Chapter 1: X-Ray Micro-Focus Monitoring of Water Alternating Gas Injection in Heterogeneous Berea Sandstone Samples

Jeffery S. Luo, Xiongyu J. Chen, D. Nicolas Espinoza, and Quoc P. Nguyen

1.1 Introduction

Water alternating gas (WAG) injection is commonly conducted in enhanced oil recovery of depleted reservoirs. However, gas flow often associates with fingering due to high gas mobility, which leaves a large portion of the reservoir unswept (Christensen et al, 1998; Christie et al., 1993). Many mobility control agents have been tested in the field and the laboratory. The agents are aimed at decreasing the permeability and/or decreasing the apparent gas viscosity; examples include foams (Tsau and Heller, 1992; Martinsen and Vassenden, 1999), polymers (Gogarty, 1967; Jennings et al., 1971) and nanoparticles (Kim et al., 2016; Emrani et al., 2017). This study primarily focuses on the use of foam as a mobility control agent for conformance improvement.

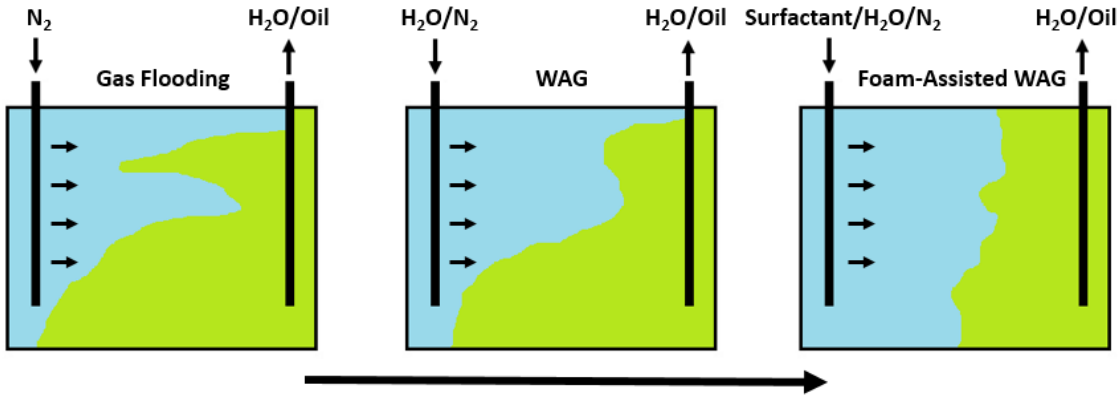

Fig. 1.1—Improved sweep efficiency, reduced effects of gravity segregation and heterogeneities in FAWAG.

Foams are commonly used in two types of applications. The first application is to improve the sweep efficiency of gas flooding through WAG injection (**Fig. 1.1**). Since many foams are shear thinning fluids, foam treatment may remain effective in conformance improvement even in far-wellbore regions due to good injectivity. The second application is to reduce gas channeling near production wells (Kovscek and Radke, 1994). The low effective density of most mobility-

control foams makes it a good candidate for selective injection treatment in the upper reservoir intervals where excessive, competing gas override is most likely to occur (Krause et al., 1992). This is especially applicable to reservoirs in which gas coning or cusping is a problem due to high vertical permeability.

Both laboratory experiments and fluid flow simulation help understand the physics of gas mobility control during WAG processes. Micro-fluidic experiments show snap-off and reform of gas bubble when passing through pore throats in 2D models. X-ray CT experiments confirm capillary or residual trapping of CO_2 is influenced by fluid and interfacial physics at the pore scale and plays an important role in subsurface CO_2 storage (Krevor et al., 2015). Estimates of trapped gas fraction and flowing gas fraction using CT imaging proves to be more accurate than 1D models used to interpret tracer effluent profiles (Nguyen et al., 2009). X-ray CT images taken during nitrogen-foam propagation showed that the mobility-reduction factor increased with surfactant concentration and total injection velocity (Simjoo et al., 2013).

This study uses X-ray microfocus radiography and coreflood experiments to measure the displacement patterns and saturations during WAG process with and without surfactants in Berea sandstone samples. The advantage of microfocus X-ray observation over medical X-ray is that it can resolve measurements at a higher resolution. The resolution of the images permits observation of not only core scale fingering but also pore-scale displacement. We use foam as a mobility control agent to decrease the apparent viscosity of injected nitrogen gas. This study also reports compositional reservoir numerical simulations to investigate and match experimental data. The ultimate goal is to better understand WAG processes in heterogeneous media by quantifying displacement patterns and saturations through both experimental and numerical modeling techniques.

1.2 Materials and Methods

1.2.1 Rock and Fluid Types

We use Berea sandstone with a permeability of ~100 mD and porosity of 17% in this study. Samples are rectangular slabs with a length of 10 cm, a height of 5 cm, and a width of 0.5 cm. The width is one order of magnitude less than the length and the height, so that changes of properties perpendicular to the sample are minimal. **Fig. 1.2** shows two examples of the sandstone slabs. The top sample has horizontal bedding direction parallel to flow. The bottom sample has vertical bedding direction perpendicular to the flow direction. Bedding was assumed to only consist of coarse sandstone (rock type 1) and tight sandstone (rock type 2) for the heterogeneous rock model. For WAG experiments, 10 wt% NaBr brine solution is used as the liquid phase, nitrogen is used as the gas phase, and 0.1% – 0.2% Bioterge AS-40 surfactant is used to create a stabilized foam.

Fig. 1.2—Berea sandstone samples.

1.2.2 Flow Automation and Data Logging

1.2.2a Experimental Setup

Fig. 1.3 shows the schematic set-up for the experiment. A sandstone slab is epoxied inside a polycarbonate frame with two inlet connections and one outlet connection. The sample is leak tested to 200 psig. Diffusers on the inlet and outlet surface of the slab ensure flow into and out of the slab is as uniform as possible. For the inlet connections, gas is injected through one connection via a gas cylinder charged to 150 psig. Water is injected through the other connection via a syringe pump designed from an open source project by Naroom (2014). Details are included in Appendix C. **Fig. 1.4** shows an X-ray radiography of the epoxied sample, which shows the rock slab, the diffusers, and the connections.

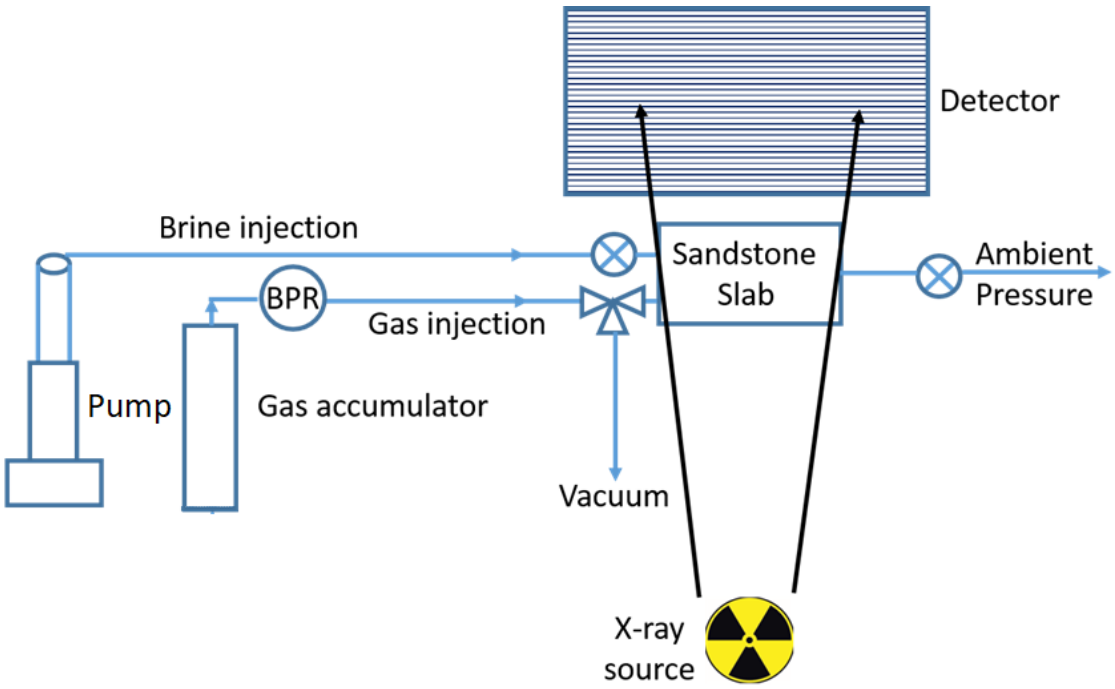

Fig. 1.3—Fluid injection schematic setup.

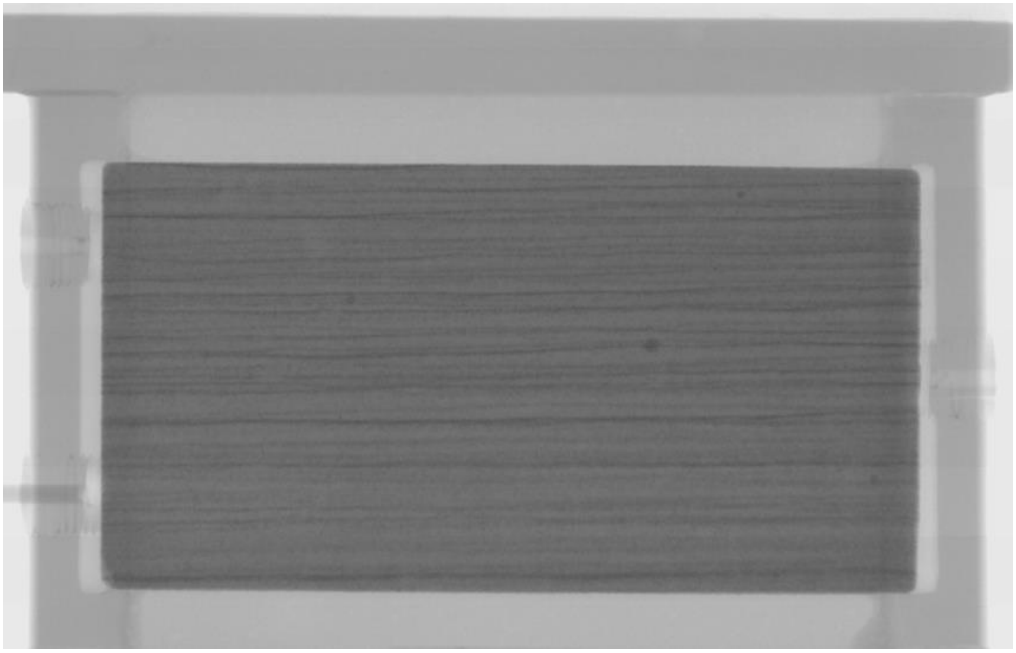

Fig. 1.4—X-ray radiography of epoxied 2D sandstone slab.

During the injection period, X-ray radiography images are taken to record the in situ saturation of the sample. Upstream and downstream pressures are automatically recorded using pressure transducers. A Bluetooth device is used to open and close valves and pumps since once scanning commences and the interlocks of the X-ray system are secured, the system cannot be manipulated physically.

To create an automated fluid injection system that is capable of remote control, an Android phone or computer is used to wirelessly operate the fluid injection apparatus via an Arduino Uno and HC-06 Bluetooth module (Nedelkovski, 2017). This allows the user to dynamically characterize internal structure changes in changing flow, pressure, and temperature conditions. Data logging is also included with the addition of a micro-SD breakout board and DS 3231 real time clock module (Karlsen, 2010). A circuit schematic and breadboard diagram of the system is shown in **Figs. 1.5 and 1.6**, respectively.

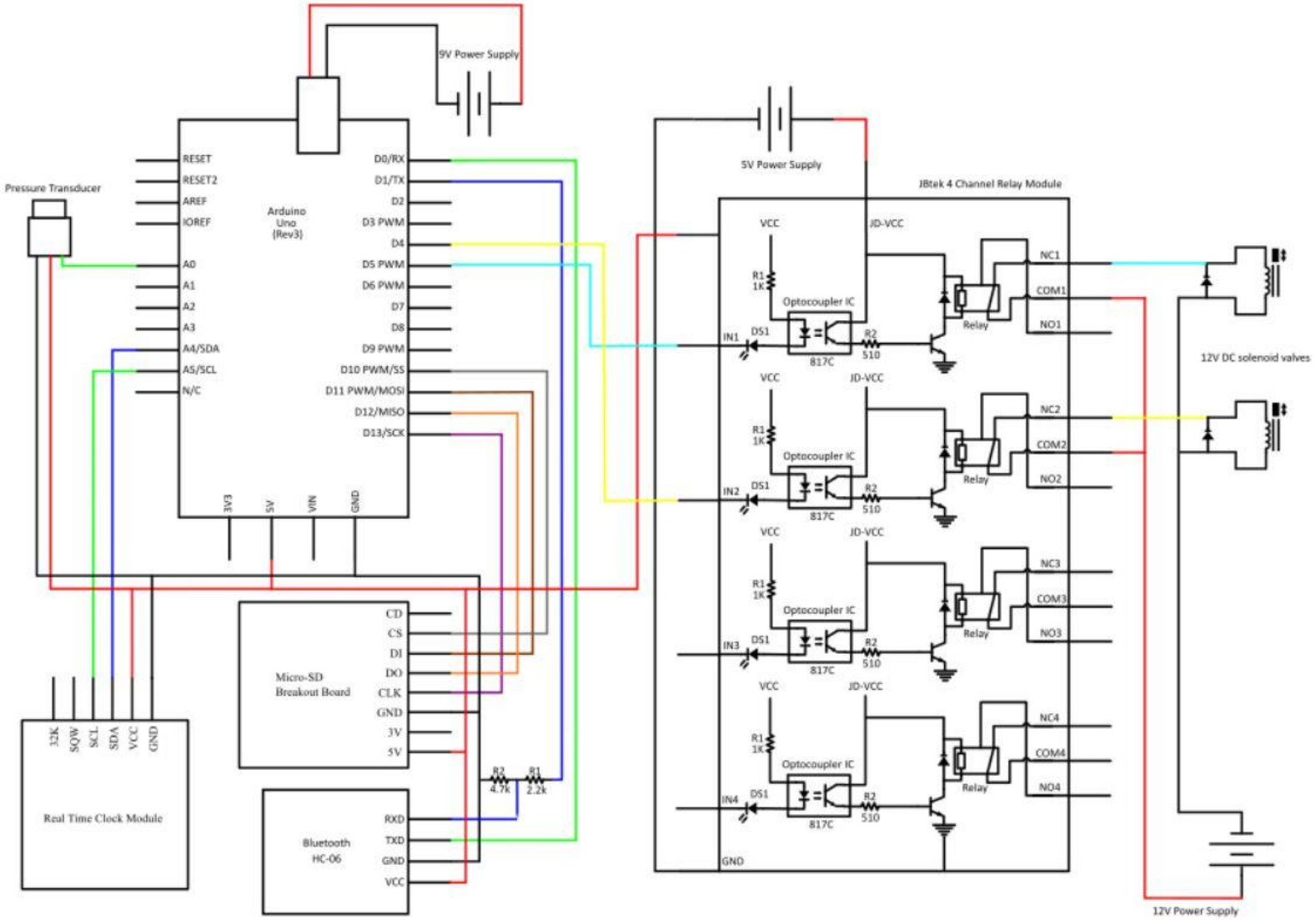

Fig. 1.5—Circuit schematic of flow automation and data logging system.

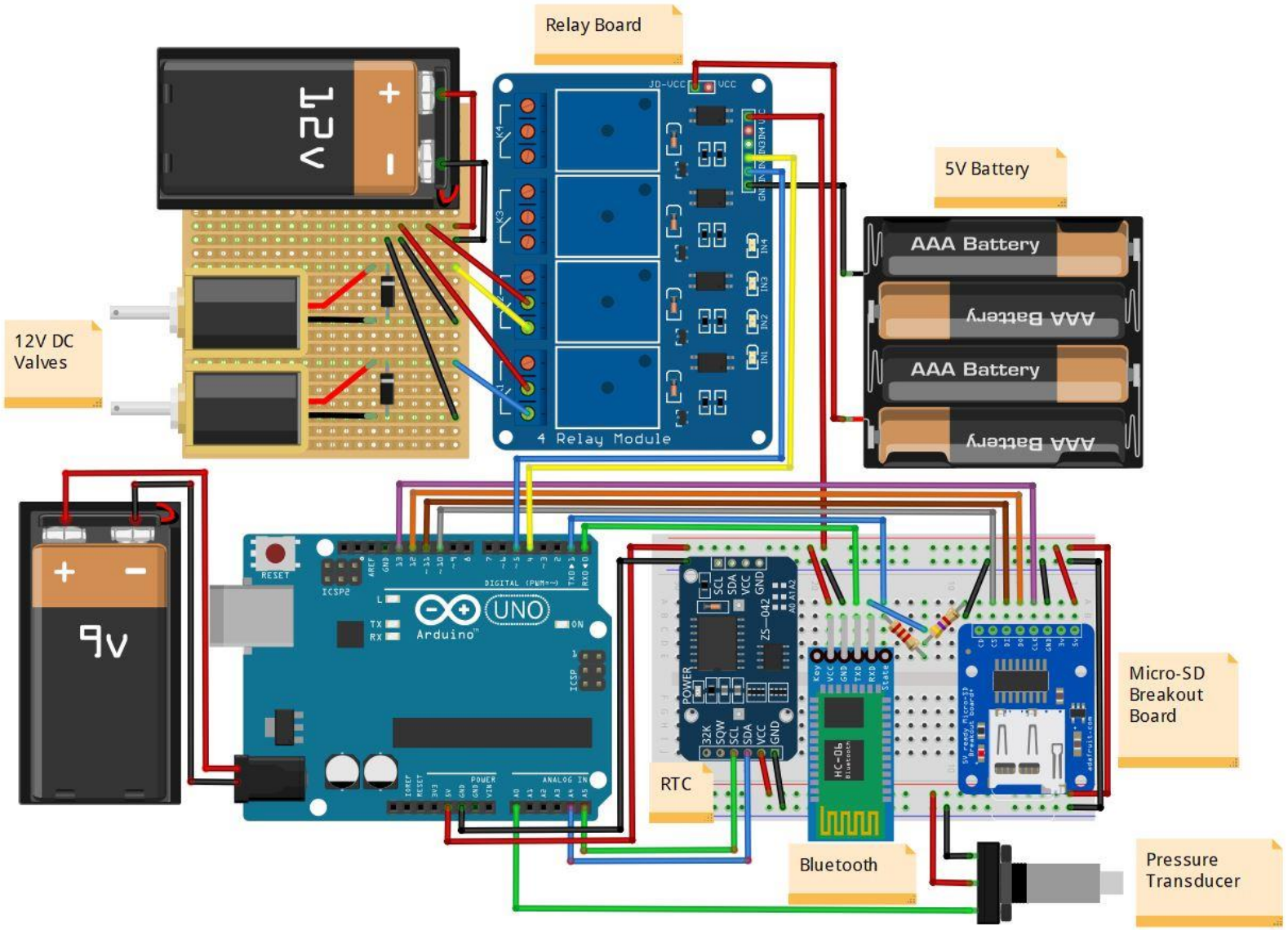

Fig. 1.6—Fritzing diagram of flow automation and data logging system.

1.2.2b Electronics for Automated System

1) Arduino Uno R3

The Arduino Uno is a microcontroller board based on the ATmega328P. A circuit schematic of the Arduino Uno is shown in **Fig. 1.7**. It has a 16 MHz quartz crystal, 6 analog inputs, 14 digital input/output pins, a USB connection port, a power jack, an ICSP header and a reset button. Power is supplied through either the USB connection port or the power jack. Since the system must be isolated, a separate 9V power supply is used to power the Arduino Uno during scanning. The pins operate at 5V with each pin capable of providing and receiving a maximum of 40 mA. Of all the specialized pins, a few are particularly relevant to this project. The serial pins of 0 (RX) and 1 (TX) receive and transmit TTL serial data, which are used to communicate with the Bluetooth module.

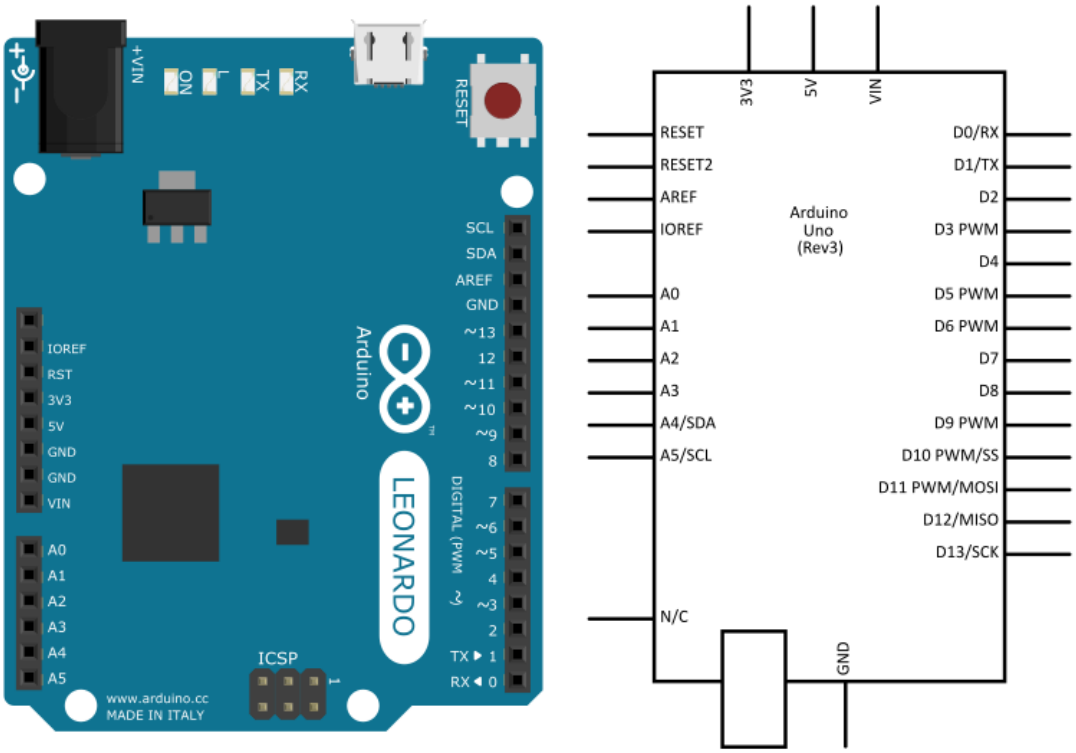

Fig. 1.7—Fritzing breadboard (left) and circuit (right) schematic of Arduino Uno.

2) HC-06 Bluetooth Module

A Bluetooth module is needed for the Android phone or computer to communicate with the Arduino Uno. An app is used to send data from an Android phone or computer to the Bluetooth module, which is connected to the Arduino Uno via a serial connection. MIT App Inventor 2 is used to create an app to control the Arduino, as detailed in Section 1.2.2d. Tera Term is used for communication between a computer and the Arduino.

The Bluetooth module has a VCC, GND, TXD, RXD, and reserve LED status output pin. The module has a current of 30 mA unpaired, 10mA paired and a working voltage from 3.3 to 6V, so the VCC pin of the Bluetooth can be connected to the 5V output pin of the Arduino for powering the module. The GND pin of the Bluetooth module can be connected to any of the GND pins on the Arduino. The TXD pin, which is used to send data from the module to the Arduino, needs to be connected to the serial receive pin (RX) of the Arduino. Similarly, the RXD pin, which is used to receive data from the Arduino, needs to be connected to the serial Arduino transmit pin (TX).

For the HC-06 Bluetooth module, the interface level for TXD and RXD is 3.3V, meaning that while the module can still be powered with 5V from the Arduino Uno, the communication lines to and from the module are only rated up to 3.3V. Sending data from the Bluetooth module to the Arduino (TXD to RX) will not be an issue since Arduino's RX line can interpret the 3.3V signal from the Bluetooth. However, when the Arduino is transmitting data to the Bluetooth (TX to RXD), the signal will be at 5V, which is higher than the working interface level of 3.3V. This may damage the module. To prevent possible damage, a voltage divider is used to decrease the voltage of the Arduino TX line from 5V to approximately 3.3V. The voltage divider equation is shown in **Eq. 1.1**, with 2.2 k**Ω** resistor as R_1, and 4.7 k**Ω** resistor as R_2. A circuit schematic of the voltage divider and Bluetooth module is shown in **Figs. 1.8 and 1.9**, respectively.

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \quad (1.1)$$

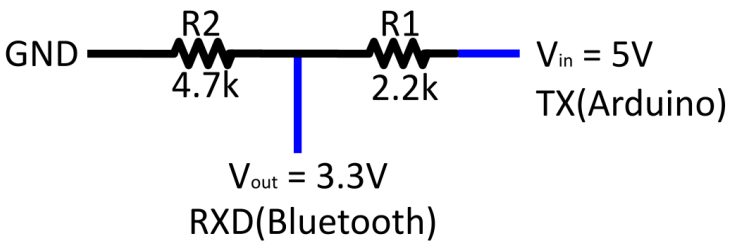

Fig. 1.8—Voltage divider schematic.

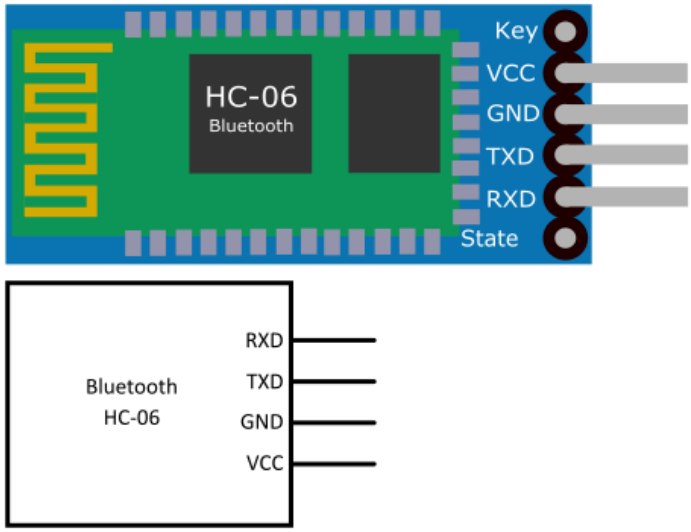

Fig. 1.9—Fritzing breadboard (top) and circuit (bottom) schematic of HC-06 bluetooth module.

Arduino source code is written to specify how the Android phone or computer will communicate with the Arduino. Refer to Section 1.2.2c for details on coding. Once the code is written, the code can be uploaded to the Arduino through the on-board port. This process, however, requires the TX and RX lines to be unplugged since the Arduino uses the serial communication lines during uploading so the pins RX (digital pin 0) and TX (digital pin1) are busy. Once the code is uploaded, the Bluetooth must be paired with the Android phone during first time use. The default password for the HC-06 Bluetooth module is 1234.

3) Jbtek 4 Channel Relay Module

This 5V relay board is used to control the 12V DC solenoid valves used for gas and water injection. Electrically operated through an electromagnet, a relay behaves as a switch that can be activated with a low voltage signal, for example 5 volts from an Arduino microcontroller.

The JBtek 4 Channel DC 5V Relay Module has 4 relays with a maximum rating of 10A at 250V AC and 10A at 30V DC. The high voltage output connector for each relay has 3 pins. The common pin (COM) is located in the middle, as highlighted in blue as shown in **Fig. 1.10**. From the markings, the other two pins are normally closed (NC) and normally opened (NO). When the relay is off during its default state, COM is connected to NC. When the relay is turned on, COM connects to NO. The solenoid valves are connected to NC and COM since the valves should be first closed upon powering of the relay. Only when the relay is triggered on will there be a common connection allowing supply to be provided to the load.

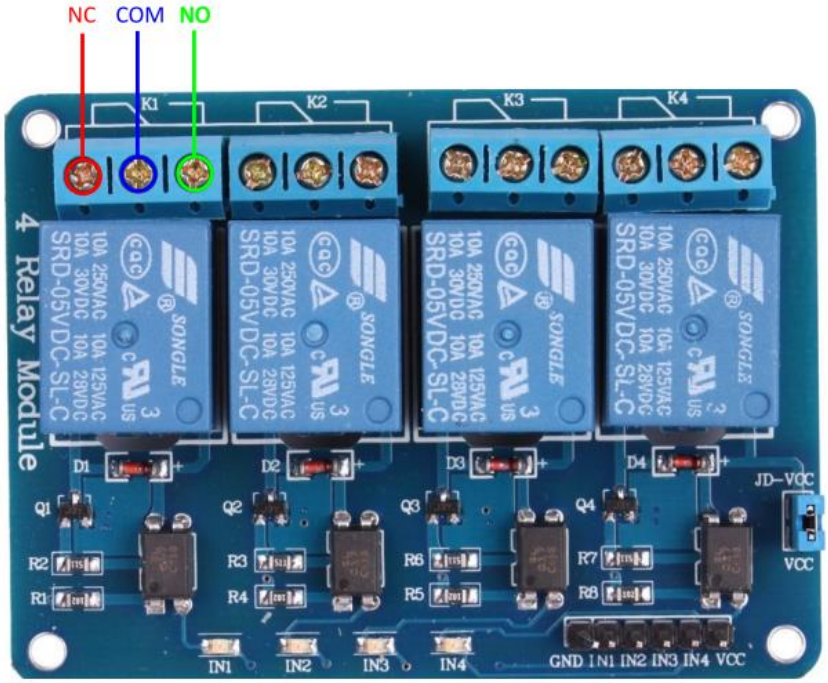

Fig. 1.10—Picture of JBtek 4 Channel DC 5V Relay Module.

On the opposite side of the module, there are 2 sets of pins. The first set has 5 pins, a ground (GND), a power pin (VCC), and 4 input pins labeled IN1 through IN4. The second set has 2 pins with a jumper between the JD-VCC and VCC pin. A circuit schematic of the relay module is shown in **Fig. 1.11**.

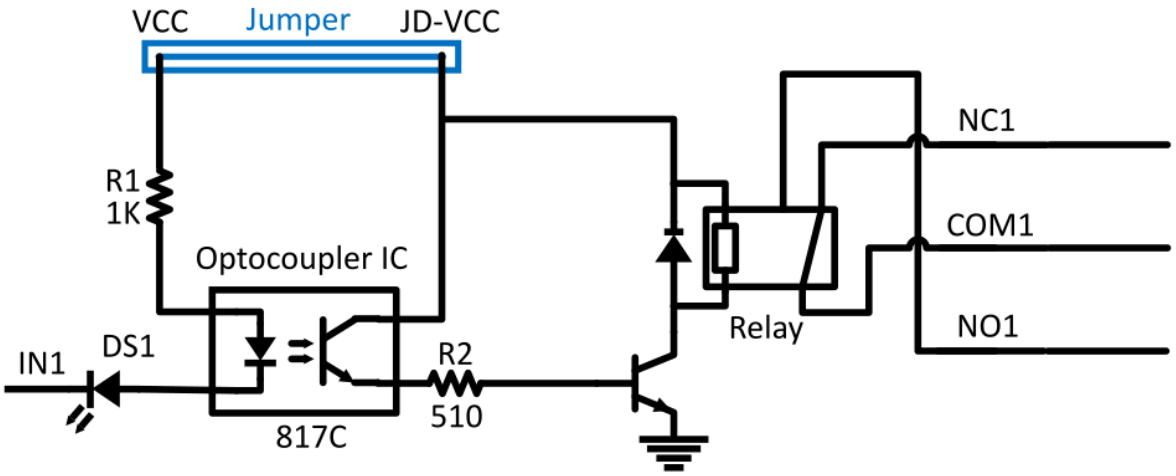

Fig. 1.11—Circuit schematic of JBtek Relay Module with jumper intact.

With the current configuration, the Arduino board directly powers the electromagnet of the relay since the 5V from the microcontroller directly feeds into both the VCC and JD-VCC pin. The VCC pin is responsible for activating the relay through the Optocoupler IC while the JD-VCC pin powers the electromagnet of the relay. No isolation between the relay and the microcontroller is provided. This poses several issues. The first being that the Arduino's 5V regulator can be over-stressed as the relay coils draw current themselves. The second issue involves the potential of a back-EMF spike that can cause the Arduino to malfunction. To circumvent these possible complications, an alternative configuration with a separate 5V regulator is used to suppress the noise and isolate the Arduino microcontroller from the relay. As shown in **Fig. 1.12**, this is accomplished by removing the jumper pin and connecting a separate 5V power supply to the exposed JD-VCC pin and the ground pin.

Now with this configuration there is no physical connection between the microcontroller and relay. Only the LED light of the Optocoupler IC is used to activate the relay.

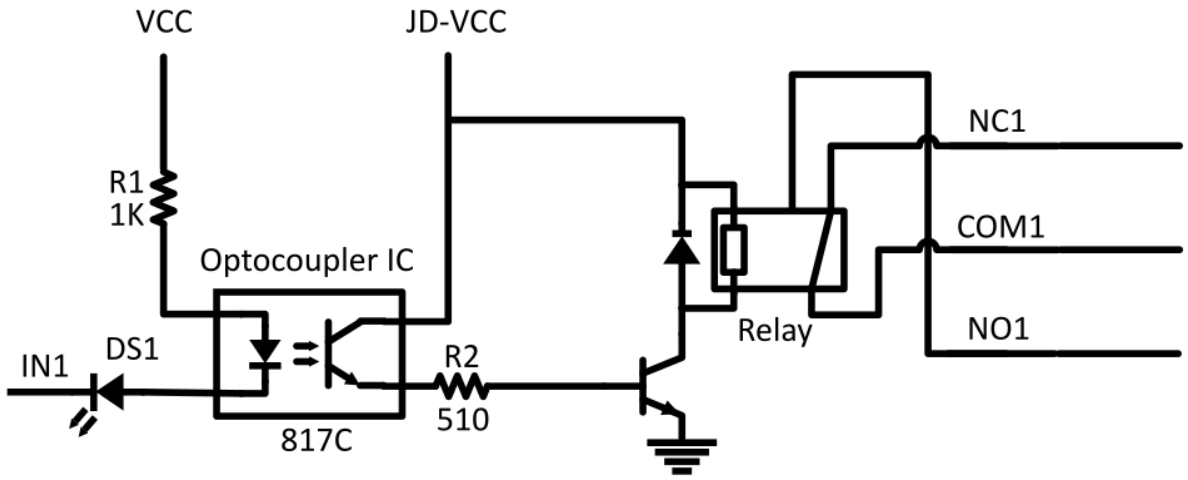

Fig. 1.12—Circuit schematic of JBtek Relay Module with jumper removed.

4) 12V DC Solenoid Valves

HFS 12V DC electronic solenoid valves are NC (normally closed), meaning they're closed by default until power is supplied. The valves have a working pressure of 0-140 psi, working temperature range of 23°F to 176°F, and orifice of 20mm. A Fritzing breadboard diagram and circuit schematic of the solenoid valve is shown in **Fig. 1.13**. A circuit schematic showing how the solenoid valves are connected to the relay board is shown in **Fig. 1.14**.

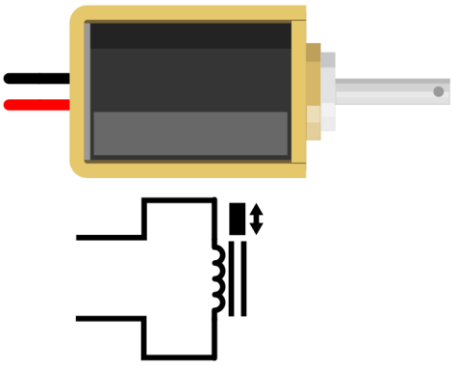

Fig. 1.13—Fritzing breadboard (top) and circuit (bottom) schematic of solenoid valve.

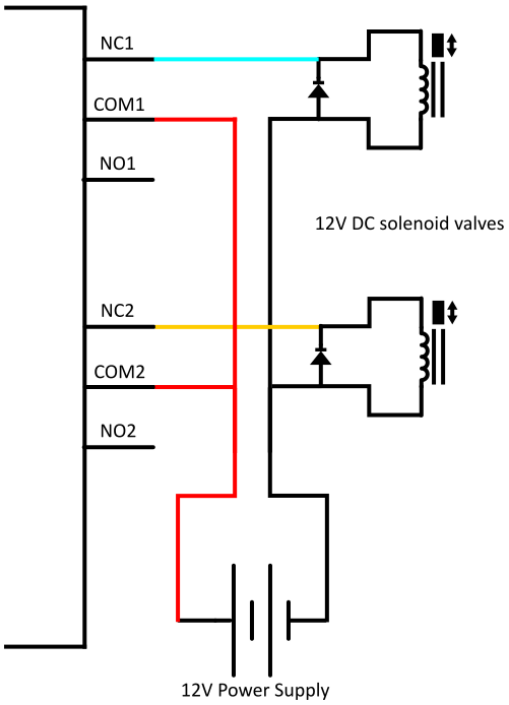

Fig. 1.14—Circuit schematic of solenoid valve setup.

The above schematic shows one end of the solenoid grounded, the other connected to the control voltage. The ground end is common to all solenoids, but the control end is connected to their respective relay, which is connected to the positive side of the 12V DC power supply. Reversed diodes are also hooked parallel to each solenoid valve. Diodes are used to suppress the high voltage back-EMF spike when the solenoid valve is switched off by limiting current flow to only one direction. Fritzing breadboard diagram and circuit schematic of a diode is shown in **Fig. 1.15**. The cathode end (band side) is connected to the control voltage side of the solenoid and the anode end is connected to the ground side of the solenoid.

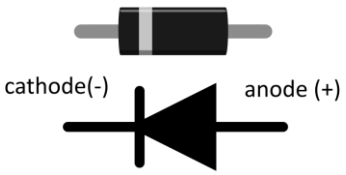

Fig. 1.15— Fritzing breadboard (top) and circuit (bottom) schematic of diode.

5) Autex Pressure Transducer

Pressure transducers in this project are used for dual purposes. First, for monitoring of upstream pressures during fluid injection. Second, for calculation of the sandpack permeability using unsteady-state and steady-state measurement type, as detailed in Section 1.2.5. The Autex pressure transducer has a working pressure range of 0 to 100 psia and linear voltage output from 0.5 V to 4.5 V. As shown with **Fig. 1.16**, the pressure transducer has 3 pins: a ground (GND), power (VCC), and output (OUT) pin. Pressure transducer was calibrated (**Fig. 1.17** and **Table 1.1**) with a maximum total error of 1.8%.

Fig. 1.16—Pictures (left and middle) and Fritzing breadboard (right) schematic of pressure transducer.

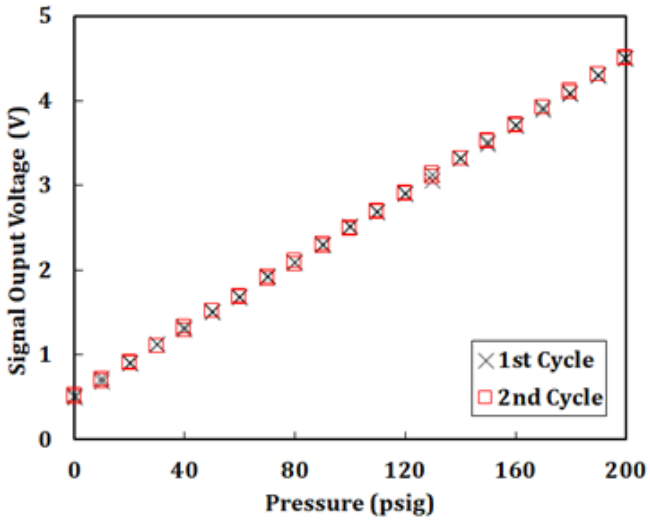

Fig. 1.17—Pressure transducer calibration.

Table 1.1—Pressure transducer error from calibration.

Calibration Type	Error Type	Error
1st Loading	Linearity	1.45%
2nd Loading	Linearity	1.62%
1st Unloading	Linearity	1.71%
2nd Unloading	Linearity	1.59%
Both Loading	Repeatability	1.72%
Both Unloading	Repeatability	1.79%
1st Cycle	Hysteresis	1.69%
2nd Cycle	Hysteresis	1.68%
Both Cycles	Total	1.80%

6) Adafruit Micro-SD Breakout Board (ADA254)

Since the Arduino microcontroller has limited memory built-in storage, a micro-SD breakout broad is needed to log pressure data. The logic level of the breakout board is strictly 3.3V, however the onboard voltage regulator permits usage with 5V systems. Between the two modes to interface with SD cards, SPI mode was chosen over SDIO mode since the former is easier for any microcontroller to communicate with as only 4 pins are required. The pin connections for SPI mode are listed below:

- 5V pin to the 5V pin of the Arduino
- GND pin to the GND pin of the Arduino
- CLK (serial clock) pin to pin 13 of the Arduino
- DO (serial data out) pin to pin 12 of the Arduino
- DI (serial data in) pin to pin 11 of the Arduino
- CS (chip select) pin to pin 10 of the Arduino

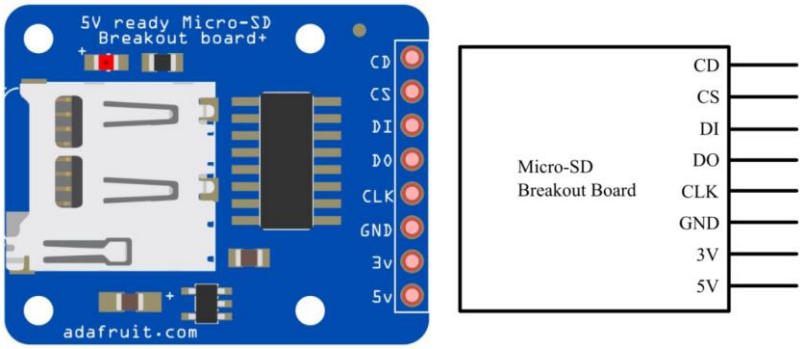

Fig. 1.18—Fritzing breadboard (left) and circuit (right) schematic of micro-SD board.

7) DS 3231 Real Time Clock Module

Since the built-in Arduino timekeeper is unreliable due to frequent power cycling of the microcontroller, a separate real time clock (RTC) module is used. Even if power for the Arduino is disconnected, the RTC module continues to keep track of time as it is powered separately. The DS 3231 RTC module shown in **Fig. 1.19** has a temperature compensated crystal oscillator for maintaining I2C real-time accuracy. It has 32K bits of memory organized as 4K by 8 bit memory. Clock accuracy is 2 ppm and error is about 1 minute per year. The module is powered through a separate 3V lithium battery (LIR2032) and has an operating voltage from 3.3V to 5V. Only four connections are needed: VCC and GND for powering the module and two I2C communication pins, SDA and SCL. For programming communication between the Arduino and RTC module, a library created by Henning Karlsen is used. The initial time is first set during first time use, which is detailed in Section 1.2.2c.

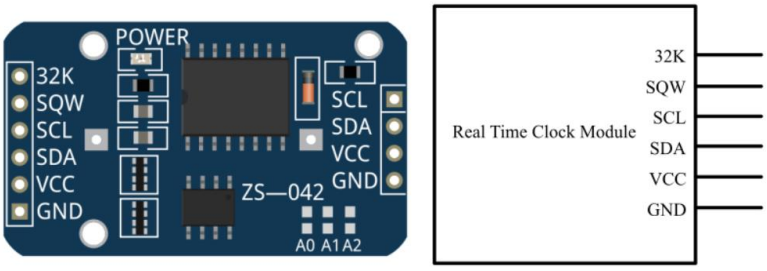

Fig. 1.19— Fritzing breadboard (left) and circuit (right) schematic of real time clock.

1.2.2c Arduino IDE for Automated System

The Arduino Integrated Development Environment (IDE) is a software coding tool that allows users to connect and communicate to the Arduino hardware through uploaded sketches. digitalWrite() and digitalRead() are all examples of functions that are used within the IDE environment to control devices connected to any of the Arduino input/output pins. For example, digitalWrite(5,HIGH) triggers a HIGH reading to whatever is connected to pin 5.

The sketch for the flow automation and data logging system, which is shown in Appendix A.1, is divided into three sections. The first section specifies all relevant variables and libraries. This includes variables like 'Received' for storing any incoming data from the serial port via Bluetooth connection and variables like 'WaterState' for monitoring the opening and closing of the water valve. The pins to which the valves are connected to are also defined in this section. This section also includes relevant libraries like <SD.h>, <SPI.h>, and <DS3231.h> for data logging. Libraries provide extra functionality for use in sketches and many come preinstalled with the IDE.

The setup section follows next. This section is called once when the sketch is first read. It is used to initialize variables, pin modes, libraries that were defined in the previous section. In this section, the baud rate or serial communication rate is set to 9600 as required by the Bluetooth module. The pin modes specified in the previous section are defined as outputs and their initial values set (i.e. water pin is defined as output and set to LOW since solenoid valves are normally closed by default). Both the SD card and real time clock are initialized in this section.

Initial time is set during first time use. This is accomplished through the functions rtc.setDOW(TUESDAY), rtc.setTime(16, 22, 0), rtc.setDate(8, 8, 2017); which sets the date of the week to Tuesday, the time to 4:22 pm in 24 hour format, and the date to August 8, 2017, respectively.

The loop section is the last part of the code. This part of the code loops continuously, allowing it to actively control the Arduino. This section is further divided into two parts, one for automated valve control and the other for data logging. For control of the valves, incoming data, when sent from the Android phone or computer via Bluetooth, is stored by the Arduino Uno as the 'Received' variable. If the character '1' is received, which is sent from the Android app when the 'WaterOn' button is pressed or from the computer via Tera Term, Arduino will open the water valve and send back the string 'WATER: ON'. Refer to Section 1.2.2d for details on the design of the Android app. Similarly, if the character '2' is received, which is sent from the Android app when the 'WaterOff' button is pressed, Arduino will close the water valve and send back the string 'WATER: OFF'. This can be repeated for the gas valve. Variables like 'WaterState' and 'GasState' are included to keep track of when the water or gas valves are closed or open.

For data logging, both time and temperature are directly called from the DS 3231 RTC after initialization. Raw pressure readings from the Autex pressure transducer range from 100 to 900, which corresponds to a pressure range of 0 to 200 psig or voltage range of 0.5V to 4.5V. The time and pressure values can be directly logged onto a file named 'myFile' SD card via functions myFile.print(rtc.getTimeStr()) and myFile.println(Pressure).

1.2.2d Android App and Tera Term for Bluetooth Communication

1) Android App

MIT App Inventor 2 is used to design a compatible app for the aforementioned Arduino sketch. **Fig. 1.20** shows a screenshot of the finished application.

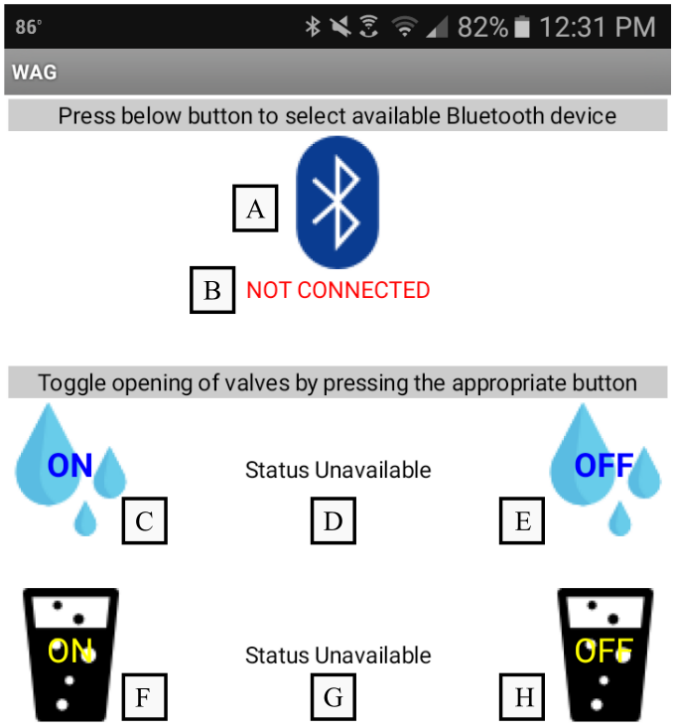

Fig. 1.20—Screenshot of 'WAG' Android application with markers A-H.

The first step to creating the application is to design the graphical user interface using the 'Designer' tab. 'HorizontalArrangements' from the 'Layout' palette is chosen and its properties (height, width, alignment) are manipulated to match the desired look. Then, buttons and labels are added to the desired 'HorizontalArrangements'. Buttons and labels are selected from the 'User Interface' Palette.

For the Bluetooth interface, from the UserInterface Palette, a 'ListPicker' is added and named 'Bluetooth_List'. A Bluetooth icon is attached as an image as shown with the marker 'A'. The ListPicker is used to select nearby Bluetooth devices to connect to. The marker 'B' is a label

named ‘Connected_Label’. This label indicates the connection status of the Android phone – whether the Bluetooth is ‘CONNECTED’ or ‘NOT CONNECTED’ to the phone. Two non-visible components are also added: the ‘BluetoothClient’ from the ‘Connectivity’ palette as well as a ‘Clock’ from the ‘Sensors’ palette, which will be used for the real time indication of the connection status.

Using the ‘Blocks Editor’ tab, blocks or function commands are assigned to the previously added components. **Fig. 1.21** shows the two blocks corresponding to the ‘Bluetooth_List’ or marker ‘A’. The top block prompts a list of paired Bluetooth devices. The bottom block allow the phone to connect to a previously selected Bluetooth address.

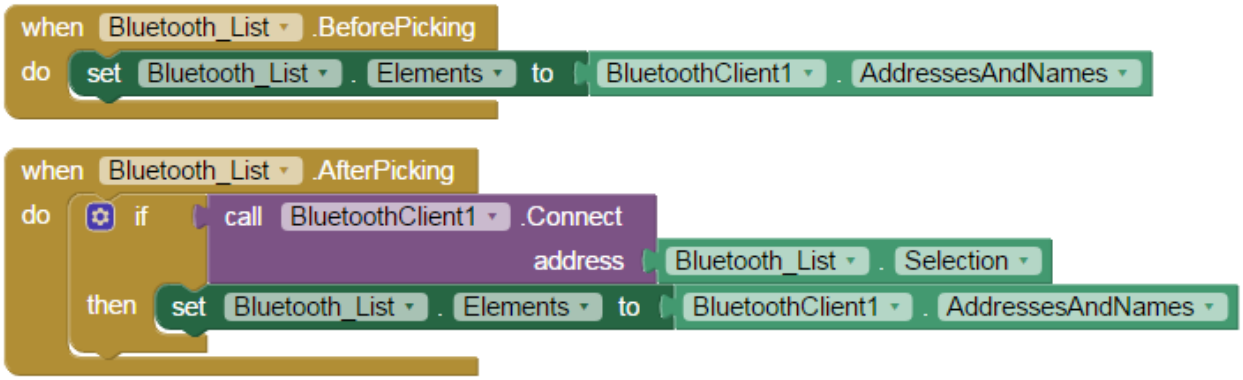

Fig. 1.21—Logic blocks corresponding to ‘Bluetooth_List’ or marker ‘A’.

Fig. 1.22 shows the block corresponding to the ‘Clock’ and ‘Connected_Label’ or marker ‘B’. This block allows for real time indication of the connection status of the Bluetooth. **Fig. 1.23** shows the Bluetooth is connected to the Android phone.

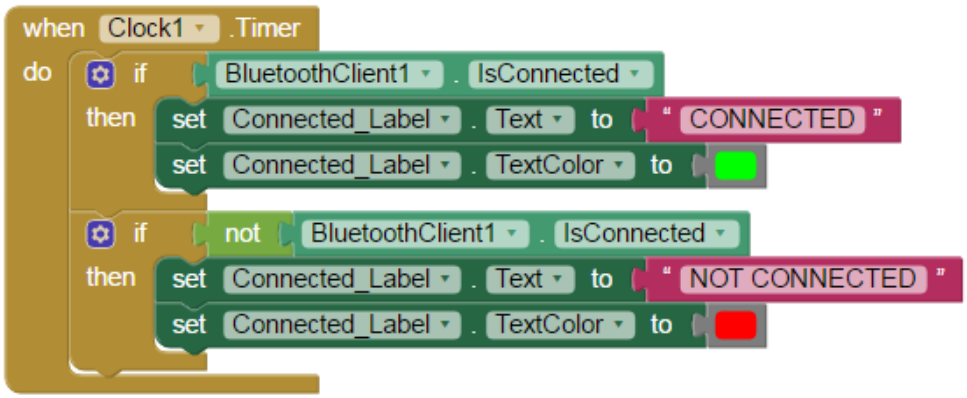

Fig. 1.22—Logic blocks corresponding to 'Clock' and 'Connected_Label' or marker 'B'.

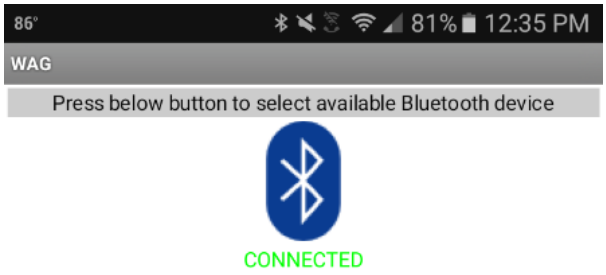

Fig. 1.23—Screenshot of 'WAG' Android application indicating the Bluetooth is activated.

The remaining markers C-H correspond to buttons and labels that send data to the Bluetooth module and Arduino Uno to either trigger the opening/closing of the respective valve or indicate the status of the valve. Like mentioned in Section 1.2.2c, incoming data, which is sent from the Android phone via the serial port, is stored by the Bluetooth module and Arduino Uno as the 'Received' variable. The incoming data are character values. Note that the 'Received' variable is an integer, so when the character '1' is sent from the Android phone, the actual value of the integer 'Received' variable is 49 according to the ASCII Table. For this app, the character '1' can only be sent when 'WaterOn' button is pressed. This button corresponds to marker 'C' of **Fig. 16**. Once this button is triggered, 'Water_Label' or marker 'D' is changed to "WATER: ON".

A respective button and label is also created for closing the water valve. The blocks corresponding to these components are shown in **Fig. 1.24**. This can be repeated for the gas valve as shown in **Fig. 1.25**.

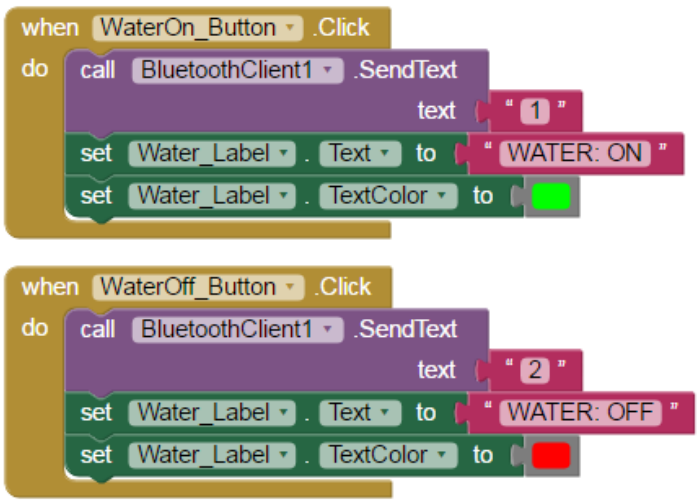

Fig. 1.24—Logic blocks corresponding to markers C-E.

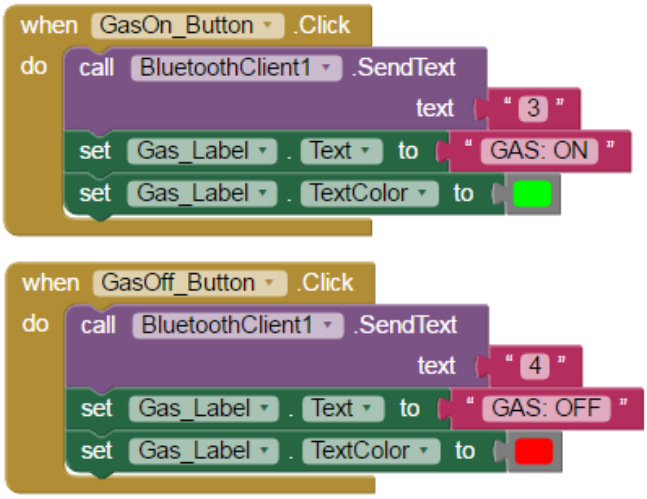

Fig. 1.25—Logic blocks corresponding to markers F-H.

2) Tera Term

Tera Term is an open-source, free terminal emulator program. **Fig. 1.26** shows how to set up a serial connection between the computer and Arduino over Bluetooth. **Fig. 1.27** shows real-time pressure measurements during an experiment.

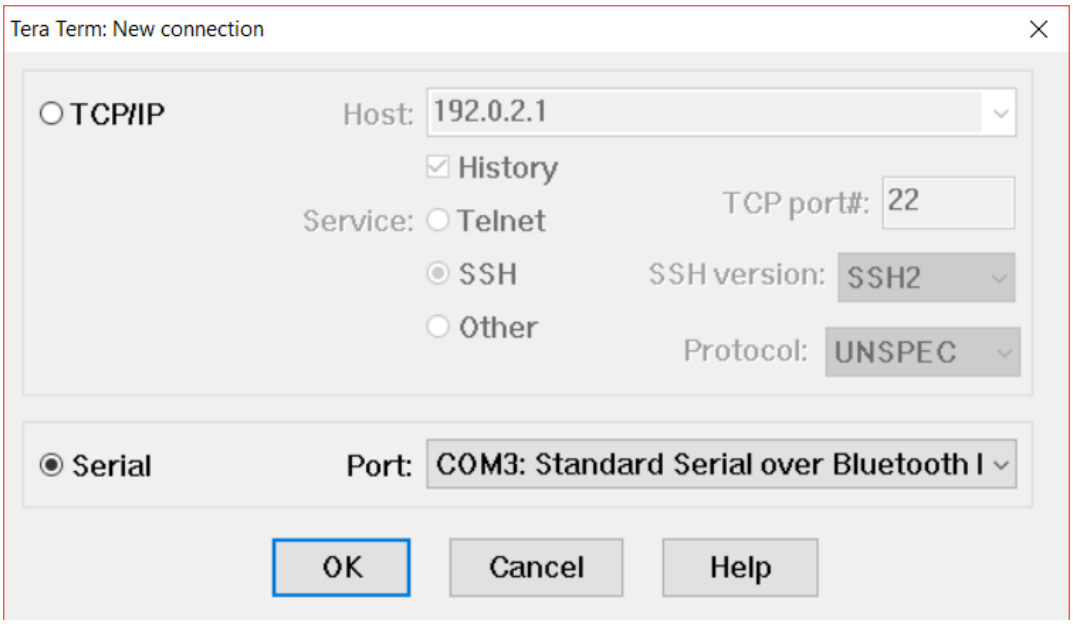

Fig. 1.26—Setting up connection between computer and Arduino via Bluetooth using Tera Term.

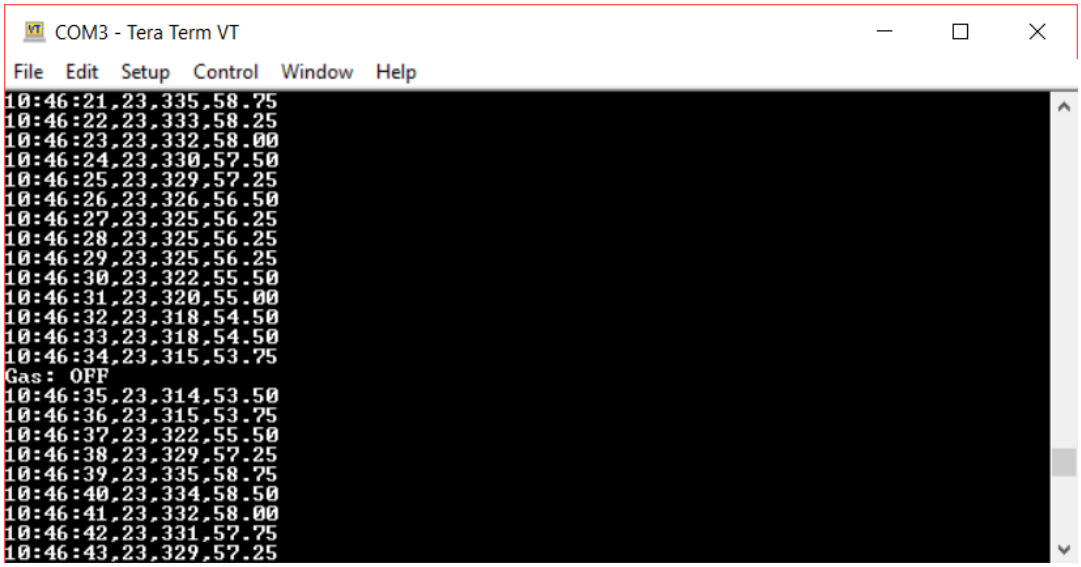

Fig. 1.27—Real-time monitoring of time-stamped pressure measurements and flow automation.

1.2.3 X-Ray System

The epoxied samples are mounted onto a Nikon XT-H 225 ST scanner. The default 225 kV microfocus X-ray source is equipped with a reflection target offering a 3 μm spot size. **Fig. 1.28** shows pictures of the epoxied rock slab sample and flow automation system within the micro-CT machine. The rock slab sample is moved on the stage to perpendicularly face the X-ray gun. As mentioned previously, the sample is regarded as 2D and hence only X-ray radiography images (not reconstructed CT images) are used for image processing and analysis.

Typical scanning parameters are a beam energy of 100 kV, beam current of 200 μA, exposure time of 1415 ms, and a resolution of 60.6 μm. The exposure time is relatively high to achieve better image quality since unlike CT images, fewer radiography images are taken throughout the experiment and there is no need for reconstruction. A copper filter with 0.48 mm thickness is used to compensate for the larger mean image brightness due to high exposure time.

One disadvantage of the radiography method is that there is no automatic flux normalization setting, which compensates for image brightness changes over time due to filament condition and system temperature. In cases where flux changes significantly over the duration of the experiment, the images must be manually calibrated – steps that are detailed in 1.3.

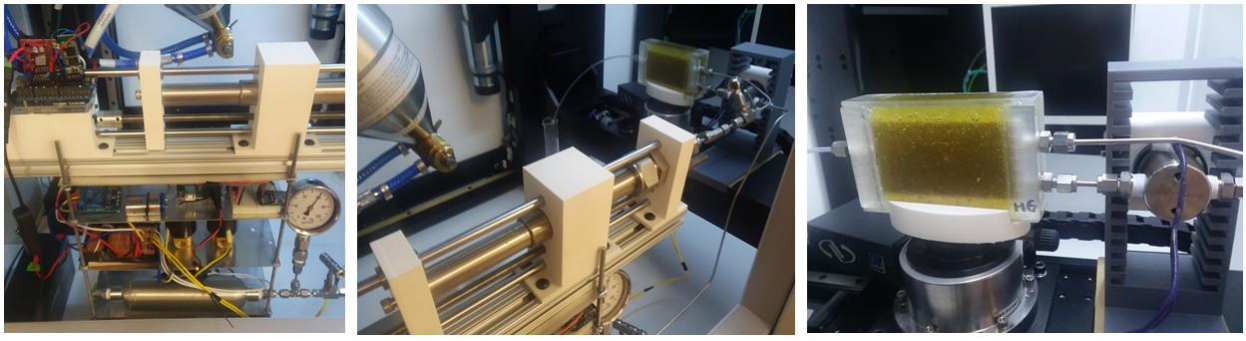

Fig. 1.28—Picture of epoxied sandstone slab and flow automation system inside X-ray micro CT machine.

1.2.4 Image Processing and Analysis

Each time a radiography image is taken an image file and XML file is outputted. The image files are in TIF format and have dimensions of 1900 by 1516 pixels. Each pixel has a grayscale value ranging from 0 to 65535 which is linearly dependent on material density and X-ray power. For example, the background air has the lowest density and thus has the highest grayscale value of 65535. During drainage, we expect an increase in grayscale values associated with the rock as nitrogen replaces the brine-filled pores. Conversely, during imbibition when the pores are once again re-saturated with brine, we expect a decrease in grayscales values associated with the rock.

The accompanying XML files include important information such as X-ray conditions, manipulator position, number of frames averaged, and the time at which the image was taken. Microsoft Excel is used to process a bundle of these XML files, mostly to determine the time elapsed for each subsequent imbibition or drainage cycle. This is important for matching pressure data recorded by the data logging system to the time at which the images were actually taken.

WAG experiments consist of alternating ~2 minute drainage and ~3 minute imbibition cycles. These times were chosen on the basis of how long it takes for the sample to be drained or re-saturated with brine. Samples are first vacuumed and dried before testing. The grayscale number of the radiography image corresponding to the vacuumed sample at a specific pixel location is called GS_{dry}. The sample is saturated with 10 wt% NaBr brine solution and scanned again. The grayscale number of this image at a specific pixel location is called GS_{wet}. Upstream the sample, gas and liquid are alternately injected at approximately 1 cm^3/min, during which the sample is scanned continuously. Downstream the sample, the injected effluent is vented to ambient pressure. The radiography image during this scanning period has a corresponding grayscale number of GS_x at a specific pixel location. Water saturation (S_w) is estimated using a simple linear relationship:

$$S_w = \frac{GS_x - GS_{dry}}{GS_{wet} - GS_{dry}} \quad (1.2)$$

MATLAB is used to create water saturation maps to visualize changes in front movement during drainage and imbibition. This is especially useful in visualizing fingering and the effect of bedding orientation on saturation pattern. Raw image files are read, cropped to include only relevant information, and filtered through a 2d median 6 x 6 filter, meaning each output pixel contains the median value of a 6-by-6 neighborhood around the corresponding pixel in the original image. Median filter improves the image quality and reduces the noise. However, too much filtering may inadvertently remove too much information and reduce image quality. Some relevant MATLAB code is included in Appendix B.

MATLAB is also used to create a binary model from the radiography images. The greyscale values of any radiography image is binarized by replacing all values above a globally determined threshold with 1s and setting all other values to 2s (**Fig. 1.29**). The tight sandstone rock corresponds to 2s and the coarse sandstone rock corresponds to 1s. This logic array is used for several reasons. First, monitoring the change in average water saturation or grayscale at any given pixel in this logic array allows for a more quantitative assessment of how rock heterogeneity affects WAG sweep efficiency. Second, numerical simulation requires grid block discretization and description. We use this binary model consisting of two rock types to model the rock heterogeneity in the laminated Berea sandstone samples.

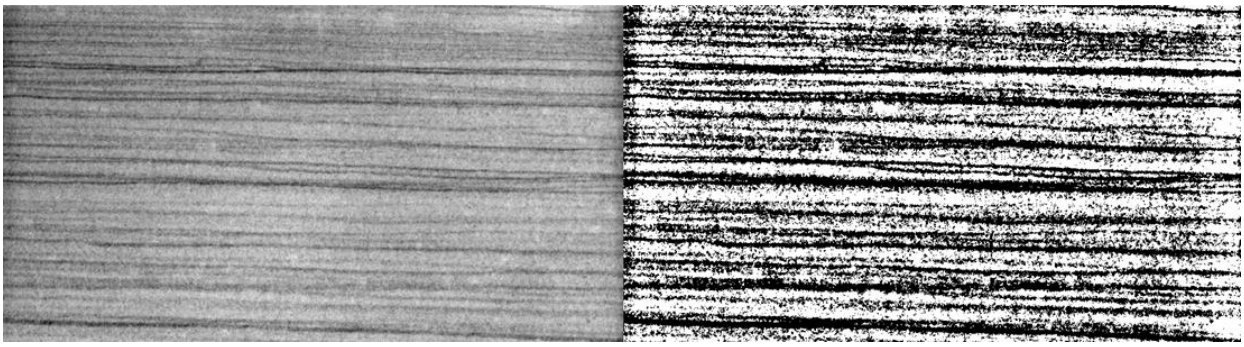

Fig. 1.29—X-ray radiography image of horizontal bedding sandstone slab (left) and corresponding binary image (right).

1.2.5 Permeability Measurement

Permeability of rectangular slab samples is measured with both unsteady-state and steady-state measurement type. Measuring the flow of pressurized nitrogen from the upstream accumulator through the epoxied slab permits the recording of a pressure transient. Layer permeabilities of the coarse sandstone (rock type 1) and tight sandstone (rock type 2) are calculated as well. Summary of results is shown in **Table 1.2**.

Table 1.2—Permeability values from laboratory measurements

Sample Type	Average k (mD)	Coarse Sandstone k1 (mD)	Tight Sandstone k2 (mD)
Vertical	19-34	110-132	10-20
Horizontal	70-120	112-195	12-13

1.2.6a Unsteady-State Permeability Measurement

While steady-state gas measurement at low pressures can rapidly determine the gas permeability, these measurements commonly deviate from the permeability to liquid or high pressure gas. Conducting these tests at elevated pressures up to 100 psia can mitigate these errors but at the cost of tedious rate measurements. Jones (1972) designed a Klinkenberg permeameter on the basis that pressure measurements can be made both more accurately and conveniently than rate measurements (**Fig. 1.30**).

The Klinkenberg permeameter consists of a core holder attached to a tank, separated by a valve. A pressure transducer is attached to the tank. To perform the unsteady-state measurement, the tank is pressurized to about 100 psig with nitrogen. Opening the bottom valve will allow the nitrogen to discharge and the pressure in the tank to decline, first rapidly than more slowly. All gas measurements include the Klinkenberg effect (**Fig. 1.31**).

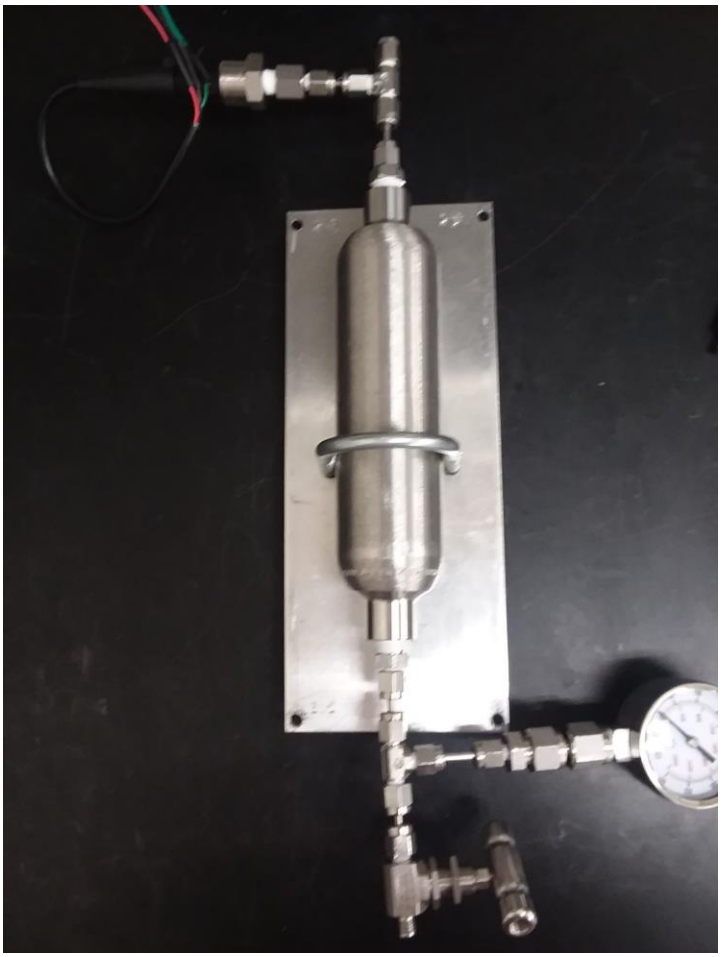

Fig. 1.30—Unsteady state measurement setup from Jones (1972).

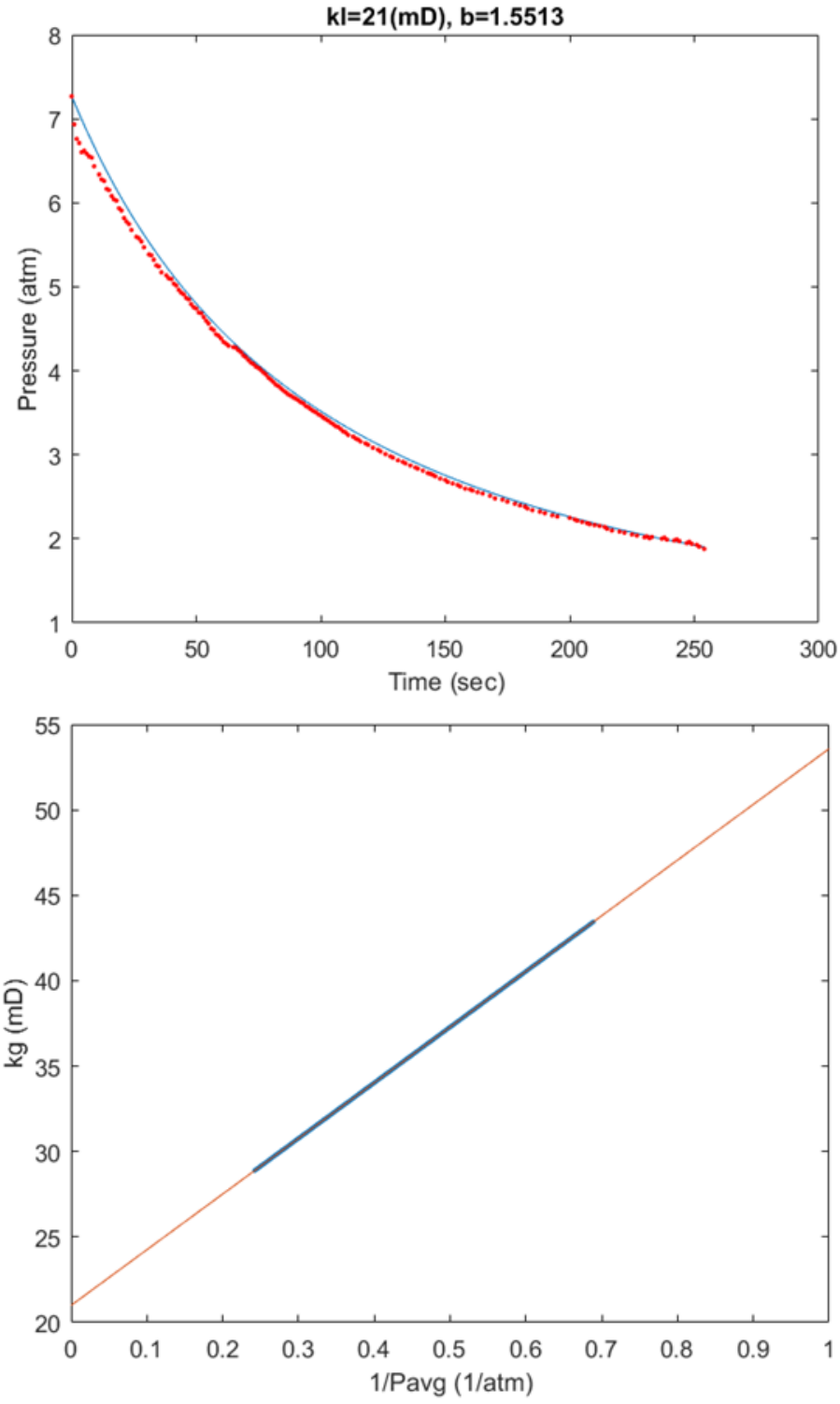

Fig. 1.31—Unsteady state pressure falloff during nitrogen injection for permeability and Klinkenberg effect correction for a vertical bedding sample.

The unsteady-state equation is derived by relating the volumetric gas flow rate at any position downstream from the inlet face of the core and Klinkenberg' s relationship to the Darcy equation. The volumetric isothermal flow rate of nitrogen in mL/sec at the inlet face of the core, q_0, is given by **Eq. 1.3** in which M is the molecule weight in gm/gm-mole, ρ_0 is the gas density at the inlet face of the core in gm/mL, n is the number of gram moles of nitrogen in the tank at time t, V_t is the volume of the tank in mL, R is the gas law constant of 82.05 ml-atm/gm-mol-K, T is the temperature in K, and P_0 is the pressure at the inlet face of the core in atm. Nitrogen is assumed to behave as an ideal gas.

$$q_0 = \frac{M}{\rho_0}\left(-\frac{dn}{dt}\right) = \frac{-MV_t}{\rho_0 RT}\left(\frac{dP_0}{dt}\right) \tag{1.3}$$

Since density is given by **Eq. 1.4**, Eq. 1.3 can be re-written as **Eq. 1.5**.

$$\rho_0 = \frac{MP_0}{RT} \tag{1.4}$$

$$q_0 = -\frac{V_t}{P_0}\left(\frac{dP_0}{dt}\right) \tag{1.5}$$

In any instant of time, the mass velocity is assumed to be constant throughout the length of the core. As nitrogen flows through the core, it expands in a manner defined by **Eq. 1.6** in which q_x is the volumetric flow rate in mL/sec that varies with both time t in seconds and distance x in cm. Substituting Eq. 1.6 and Klinkenberg's relationship (**Eq. 1.7**) into the Darcy equation for one-dimensional flow yields **Eq. 1.8**. k_x is the permeability as a point function. k_l is the Klinkenberg or liquid permeability in md, b is the Klinkenberg slip factor in atm, P_a is the atmospheric pressure in atm.

$$q_x = \frac{q_0 P_0}{P_x} = -\frac{V_t}{P_x}\left(\frac{dP_0}{dt}\right) \tag{1.6}$$

$$k_x = k_l\left(1 + \frac{\mathrm{b}}{\mathrm{P}_x + P_a}\right) \tag{1.7}$$

$$\frac{-V_t}{P_x}\left(\frac{dP_0}{dt}\right) = \frac{k_l A\left(1+\frac{\mathrm{b}}{\mathrm{P}_x}\right)}{\mu}\frac{dP_x}{dx} \tag{1.8}$$

Integrating Eq. 1.7 with respect to length, L, dividing the result by $0.5(P_l - P_0)$, and converting to field units will give **Eqs. 1.9a to 1.9c**.

$$\frac{-V_t}{P_0}\left(\frac{dP_0}{dt}\right) = i + mP_0 \tag{1.9a}$$

$$i = 2(P_a + b)m \tag{1.9b}$$

$$m = \frac{k_l A}{29390\mu L} \tag{1.9c}$$

Evaluation of left right hand side of Eq. 1.9a requires calculation of pressure time derivatives. Since this procedure is inconvenient, Eq. 1.9a can be simplified by introducing a variable named y as shown in **Eq. 1.10**. y can be evaluated at two discrete values of pressure at the inlet core face, P_1 and P_2, and the corresponding times t_1 and t_2 (**Eq. 1.11**). The simplified form of Eq. 8a is shown with **Eq. 1.12**. Values of y plotted against corresponding values of P_g should yield a straight line with slope m and intercept i. P_g is the geometric mean of pressures P_1 and P_2 as shown with **Eq. 1.13**.

$$y = \frac{-V_t}{P_0}\left(\frac{dP_0}{dt}\right) \tag{1.10}$$

$$y = \frac{V_t}{t_2 - t_1}\ln\left(\frac{P_1}{P_2}\right) \tag{1.11}$$

$$y = i + mP_g \tag{1.12}$$

$$P_0 = P_g = \sqrt{P_1 P_2} \tag{1.13}$$

1.2.6b Steady-State Permeability Measurement

Steady-state measurement type is conducted with brine. Measuring the flow of brine from the syringe pump or accumulator through the epoxied slab permits the recording of a pressure transient and rate. The recorded pressure transient and rate is analyzed using the steady state method to calculate permeability (**Fig. 1.32**).

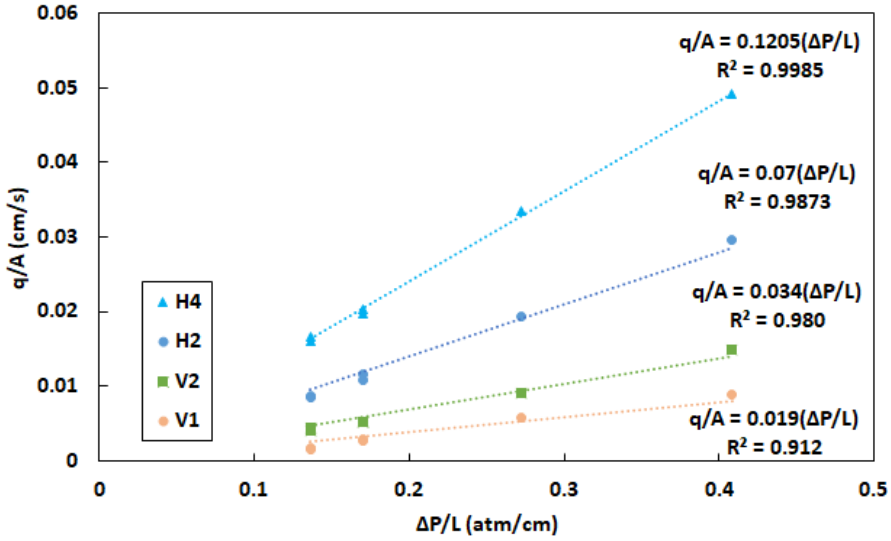

Fig. 1.32—Steady state permeability calculation method.

1.2.6c Layer Permeability Calculation

For horizontal bedding samples, the layer permeabilities of the coarse sandstone (rock type 1) and tight sandstone (rock type 2) is calculated using **Eqs. 1.14** and **1.15** where h_n is the height of the bedding layer, k_n is the permeability of the bedding layer (either k_1, the coarse sandstone permeability or k_2, the tight sandstone permeability), k_x is the horizontal permeability, and k_z is the vertical permeability.

$$k_x = \frac{\sum h_n k_n}{\sum h_n} \tag{1.14}$$

$$k_z = \frac{\sum h_n}{\sum \frac{h_n}{k_n}} \tag{1.15}$$

For vertical bedding samples, the layer permeabilities can be calculated using **Eqs. 1.16** and **1.17** where W_n is the width of the bedding layer, k_n is the permeability of the bedding layer (either k_1, the coarse sandstone permeability or k_2, tight sandstone permeability), k_x is the horizontal permeability, and k_z is the vertical permeability.

$$k_x = \frac{\sum W_n}{\sum \frac{W_n}{k_n}} \tag{1.16}$$

$$k_z = \frac{\sum W_n k_n}{\sum W_n} \tag{1.17}$$

1.3 Image Optimization and Calibration

1.3.1 Image Optimization

Optimizing a CT or radiography scan requires maximum resolution and optimized image contrast without compromising scan time. Since only radiography images are taken, a high exposure time of 1415 ms is used to maximize resolution. A copper filter with 0.48 mm thickness is used to compensate for the larger mean image brightness due to high exposure time. An even higher resolution can be achieved by moving the sample toward the X-ray source, thereby increasing the geometric magnification and decreasing the effective pixel size, but due to the size of the sample, only a resolution of 60.6 μm is achieved. Also, positioning the sample perpendicular to the X-ray source reduces the distance X-rays must penetrate, thereby enhancing image quality.

Too high of an exposure time or X-ray power setting results in an oversaturated image (**Fig. 1.33**). In the corresponding image histogram or frequency distribution of greyscale values ranging from 0 to 65535 (2^{16}), the distance between sample and background is not optimized (**Fig. 1.34**). Optimizing the distance between sample and background is important in producing sharp images with good contrast.

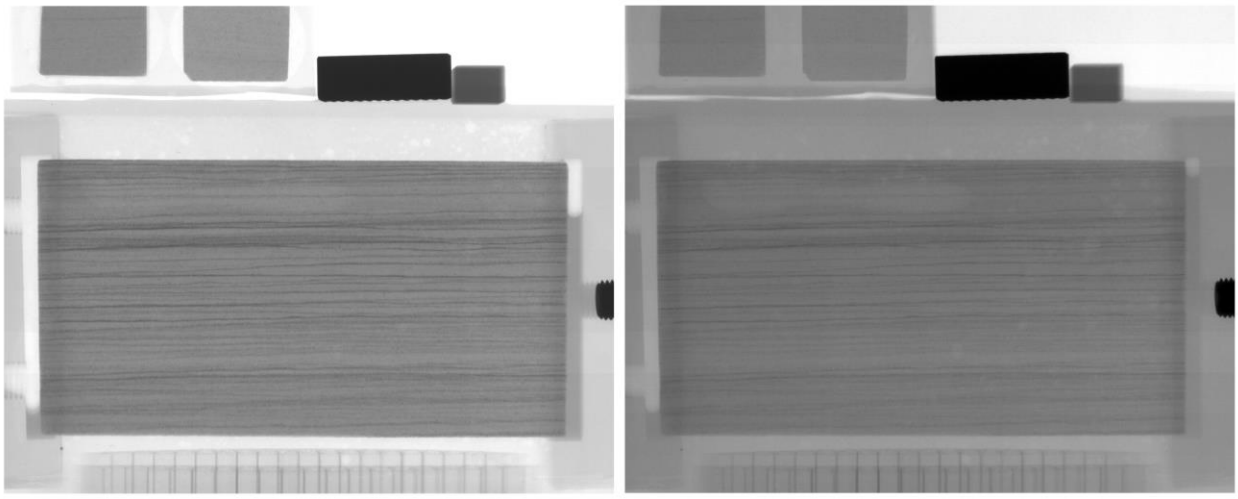

Fig. 1.33—Oversaturated image (left) versus optimized image (right).

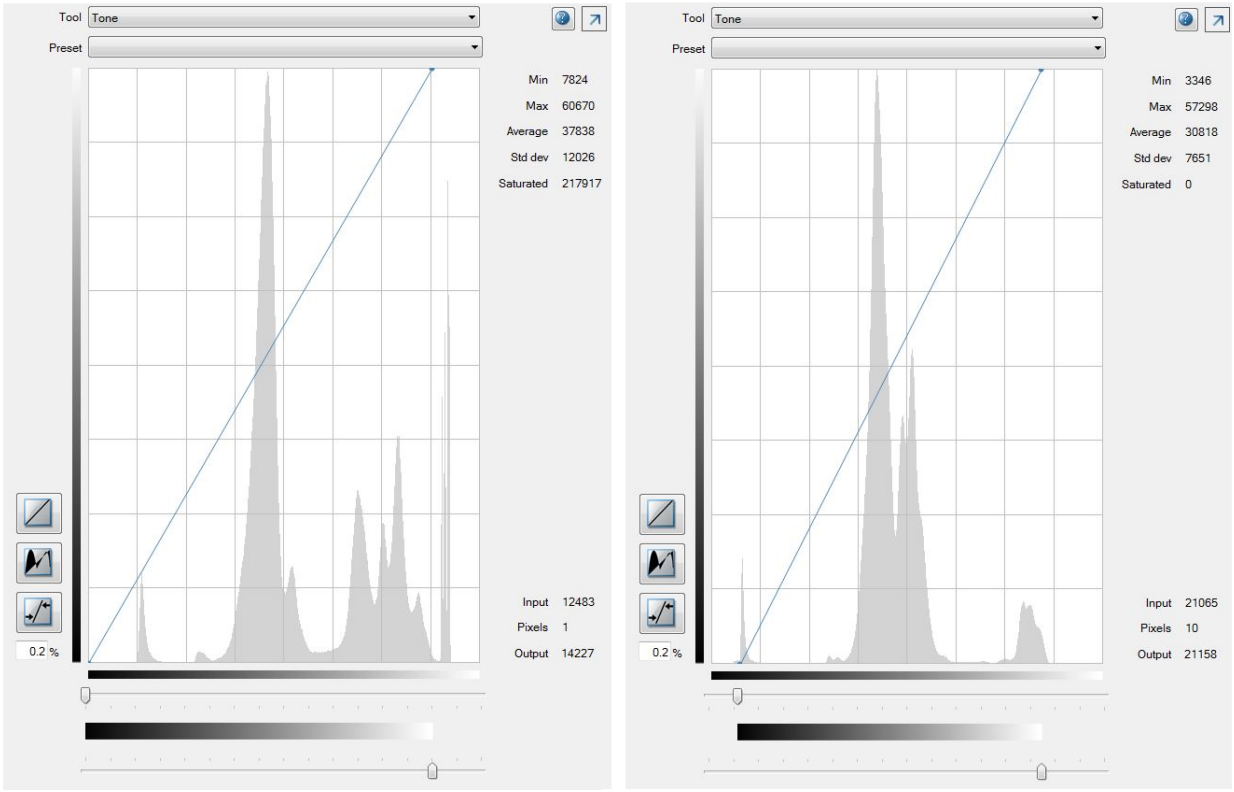

Fig. 1.34—Oversaturated image histogram (left) versus optimized image histogram (right).

1.3.1 Calibration Techniques

1.3.1a Continuous Flux Normalization

Unlike CT scanning, the radiography method has no automatic flux normalization setting to compensate for changing image brightness over time due to filament condition and system temperature. This means if the flux changes significantly over the duration of the experiment, then the greyscale values associated with the images will be offset, leading to inaccurate water saturation calculations. This usually does not pose a problem for short scans or when the filament life is in good condition, but in cases in which flux variation is significant over the course of the experiment manual calibration is needed.

The Nikon XT-H 225 ST scanner's auto-conditioning function, which ensures that the X-ray source achieves a stable operating voltage needed for steady images, reduces the magnitude of the problem. Auto-conditioning should be done on a daily basis, preferably once the machine has remained stationary for longer than 3 hours. Also, increasing the number of frames averaged for a single radiography image helps reduce the noise. The drawback of this stacking effect is fewer consecutive images can be taken during an experiment.

When manual calibration is needed, the offset in grayscale value due to unstable flux is fixed by a variable correctional factor along time. The correctional factors are determined by monitoring the change in average grayscale in certain pixel regions where we expect no change in material property. These calibration regions include the background air, aluminum plates of varying width, and epoxy regions. During experiments, the only region in which the average grayscale should be changing with time would be the rock portion since the average density of these regions are changing with time as the pores are being filled and drained with subsequent imbibition and drainage cycles.

The calibration regions monitored to determine the correctional factor is shown in **Fig. 1.35**. The dark blue region is the background air. The yellow and green region are aluminum plates of varying width. The pink, cyan, red, and black regions are different regions of the epoxy cast.

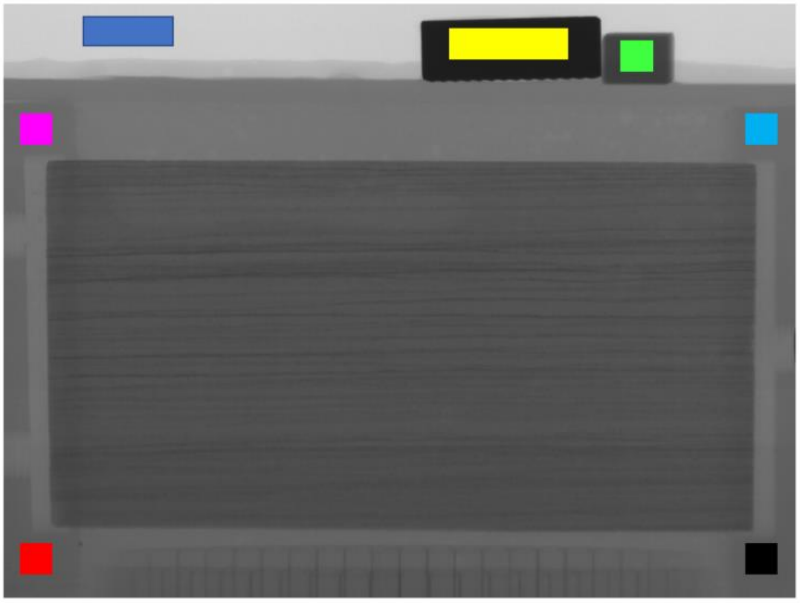

Fig. 1.35—Calibration regions used to determine correctional factor used for manual calibration.

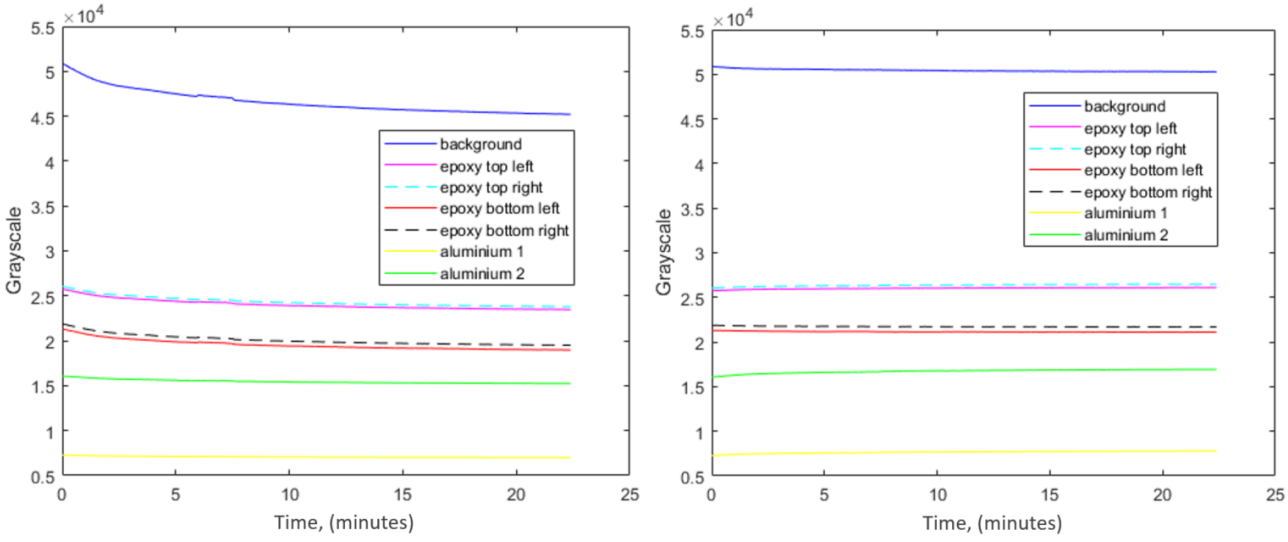

Fig. 1.36—Average grayscale number of calibration regions as a function of time before calibration (left) and after calibration (right).

Fig. 1.36 shows the average grayscale number of these regions as a function of time for a horizontal bedding sample (H3) before and after calibration. Note before calibration, there is a significant leak in flux throughout time shown with the decreasing average grayscale number. Flux leakage is especially large during the first 10 minutes of the experiment. After calibration, the average grayscale number for all calibration regions remain relatively constant throughout time. The difference in average water saturation calculation due to calibration is shown in **Fig. 1.37**. Before calibration, the offset of the grayscale number due to flux leakage leads to erroneous water saturation calculations above 100%, which is nonphysical. After calibration, the offset is corrected.

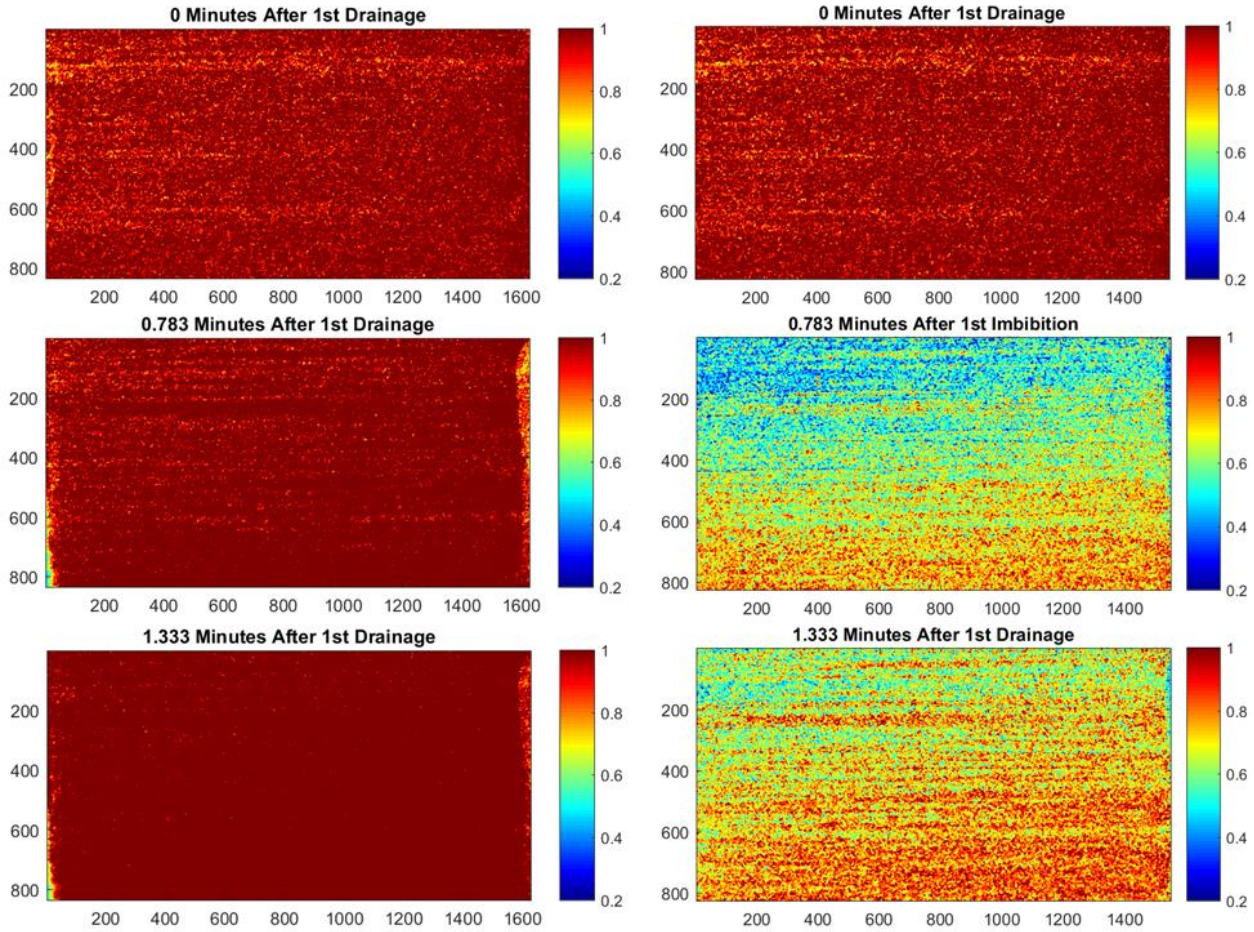

Fig. 1.37—Water saturation measurements of H3 horizontal bedding before calibration (left) and after calibration (right) during primary drainage.

Figs. 1.38 shows the manual calibration technique with another horizontal bedding sample (H5). Note flux variation is especially large during the first 10 minutes of the experiment. Average grayscale of the rock as a function of time before and after calibration is shown in **Fig. 1.39**. Before calibration, it is hard to distinguish from the plot when subsequent drainage and imbibition cycles occur. After calibration, it is much easier to distinguish when drainage occurs (increasing grayscale number) and when imbibition occurs (decreasing grayscale number) by the cyclicity of the plot.

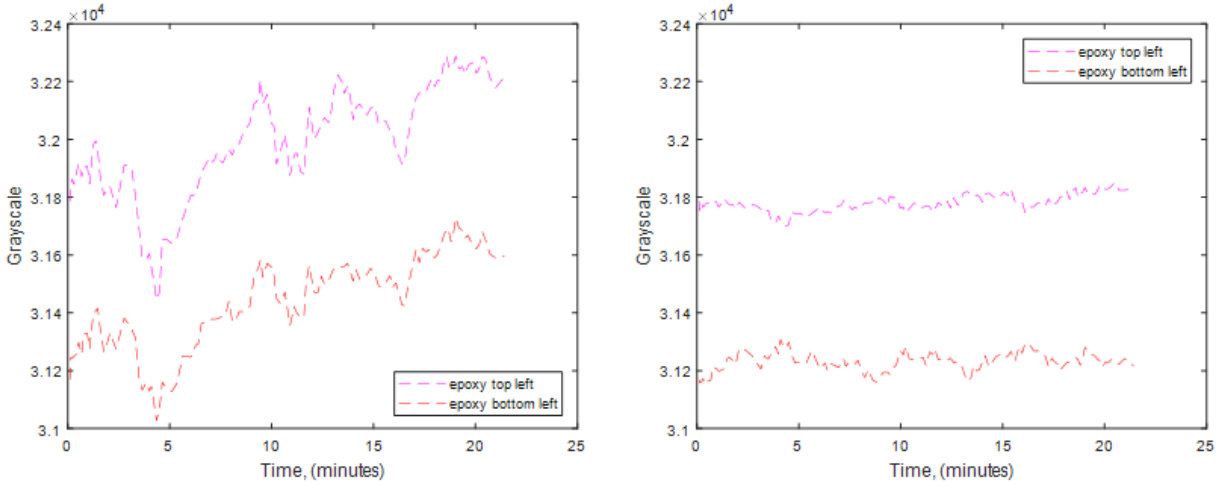

Fig. 1.38—Average grayscale number of left epoxy calibration regions as a function of time before calibration (left) and after calibration (right).

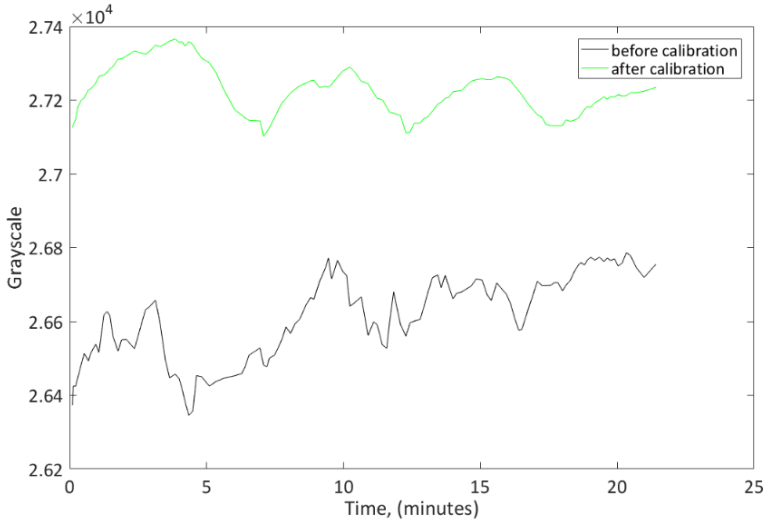

Fig. 1.39—Average grayscale number of rock as a function of time before and after calibration.

1.3.1b Initial Flux Normalization

Another recurring problem associated with radiography is the variation in flux when X-rays are power cycled (i.e. the machine is turned off and on again). Even at the same X-ray power settings, if the X-rays are temporarily turned off, the initial flux after resuming power will be different than the flux before the interruption in power. This will lead to an offset grayscale image once scanning is resumed. Sometimes the difference is significant enough that it will lead to erroneous calculations in water saturation.

This is important because as mentioned previously in Section 1.2.4, water saturation is calculated based on a linear interpolation between the image of interest to that of the dry and wet image. The dry image is the first image that is taken after the sample is vacuumed and dried. The wet image is the image taken after the sample is completely saturated with brine. This is done outside of the CT machine using a separate pump, because fully saturating the sample requires large amounts of brine injection volume that exceeds the total syringe pump volume. Fully saturating the sample from its dry state can take up to 2 hours.

Hence, there is a minimum of one instance in which the X-rays must be powered off during an experiment. During WAG injection, it may become necessary to power off the X-rays a second or third time. This usually occurs due to prematurely depleted nitrogen gas within the gas accumulator. When this occurs, scanning is paused, valves are closed, gas accumulator refilled, valves reopened, and scanning resumed. It was determined that these instances of power cycling does not have a significant impact on analysis since it usually only takes a few minutes to refill the gas accumulator and resume scanning. However, anything longer in duration increases the probability that the X-ray flux will be unstable once it is resumed.

The effect of the power cycling event can be determined by analyzing the appropriate image histograms before and after such an event. For example, for the first instance in which the X-rays are powered off so that the dry sample can be fully saturated with brine to give the wet sample, the wet image histogram should be immediately compared to the dry image histogram before continuing on with the experiment. Ideally, if the impact of the power cycling event is not severe, then subtracting the dry image histogram from the wet image histogram should give all negative grayscale values since a saturated rock sample has a higher material density and hence lower grayscale values for any given pixel region. If this is not the case, then the initial flux can be normalized by calibrating the subsequent images with an offset. **Fig. 1.40** shows the grayscale histogram of wet-dry image before and after calibration. Before calibration, wet-dry histogram shows some values of grayscale above zero. After applying the offset, wet-dry histogram shows all counts of grayscale below zero. This normalization technique minimizes non-physical, calculated water saturation values such as values exceeding 100% water saturation or negative water saturation values.

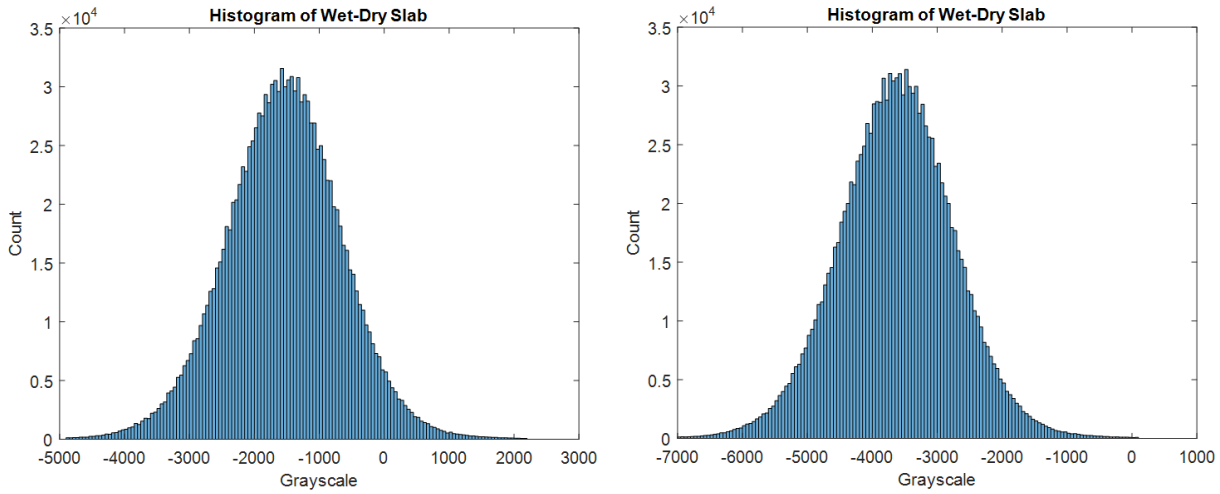

Fig. 1.40—Grayscale histogram of wet-dry image before calibration (left) and after calibration (right).

1.4 Results and Discussion

1.4.1 WAG Injection Parallel to Bedding Direction

Fig. 1.41 shows the saturation map versus time of a horizontal bedding (H4) sample during primary drainage. Fingering and early breakthrough are observed. This bedding orientation facilitates gas to flow through high permeability layers and leaves a large portion unswept in the beginning. The rock sample water saturation is evenly reduced after multiple PV of gas are injected. The laminated saturation pattern is a result of the bedding orientation. High water saturation is observed near the outlet because of the capillary end effect. **Fig. 1.42** shows the saturation maps versus time of the same sample during primary imbibition. The laminated saturation pattern is pronounced with an uneven displacement front.

Fig. 1.43 shows the average water saturation through 4 WAG cycles for both the horizontal bedding (H4) and vertical bedding sample (V2). A cycle consists of 2 minutes of drainage with nitrogen and 3 minutes of imbibition with brine. For both samples, subsequent drainage results in lower average water saturation since preceding imbibition processes usually do not fully re-saturate the sample. An exception occurs in the secondary drainage of H4.

1.4.2 WAG Injection Perpendicular to Bedding Direction

Fig. 1.41 shows the saturation map versus time of a vertical bedding (V2) sample during primary drainage. The strip-wise saturation pattern is a result of the bedding orientation. In this case, little fingering is observed. Furthermore, the displacement front is relatively uniform and sweep efficiency is high. At the end of the gas injection, high water saturation is observed near the outlet due to capillary end effect. **Fig. 1.42** shows the saturation map versus time of a vertical bedding sample (V2) during primary imbibition. The brine displacement front is even.

Fig. 1.43 shows the average water saturation through 4 WAG cycles. At any given time, the average water saturation of the vertical bedding sample is greater than the horizontal bedding sample due to the lack of fingering and a more even displacement front.

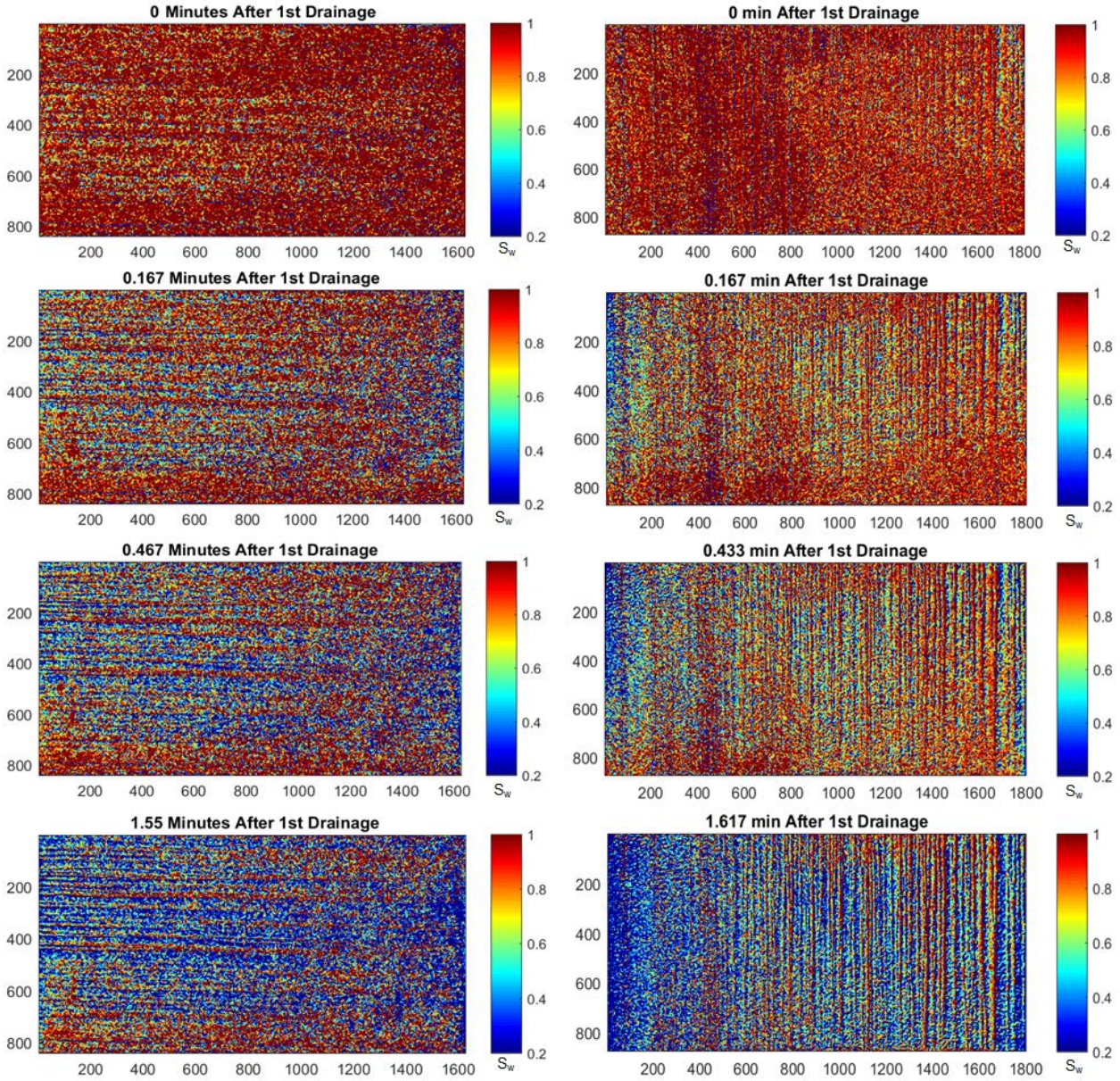

Fig. 1.41—Water saturation measurements of V2 vertical bedding (left) and H4 horizontal bedding (right) during primary drainage.

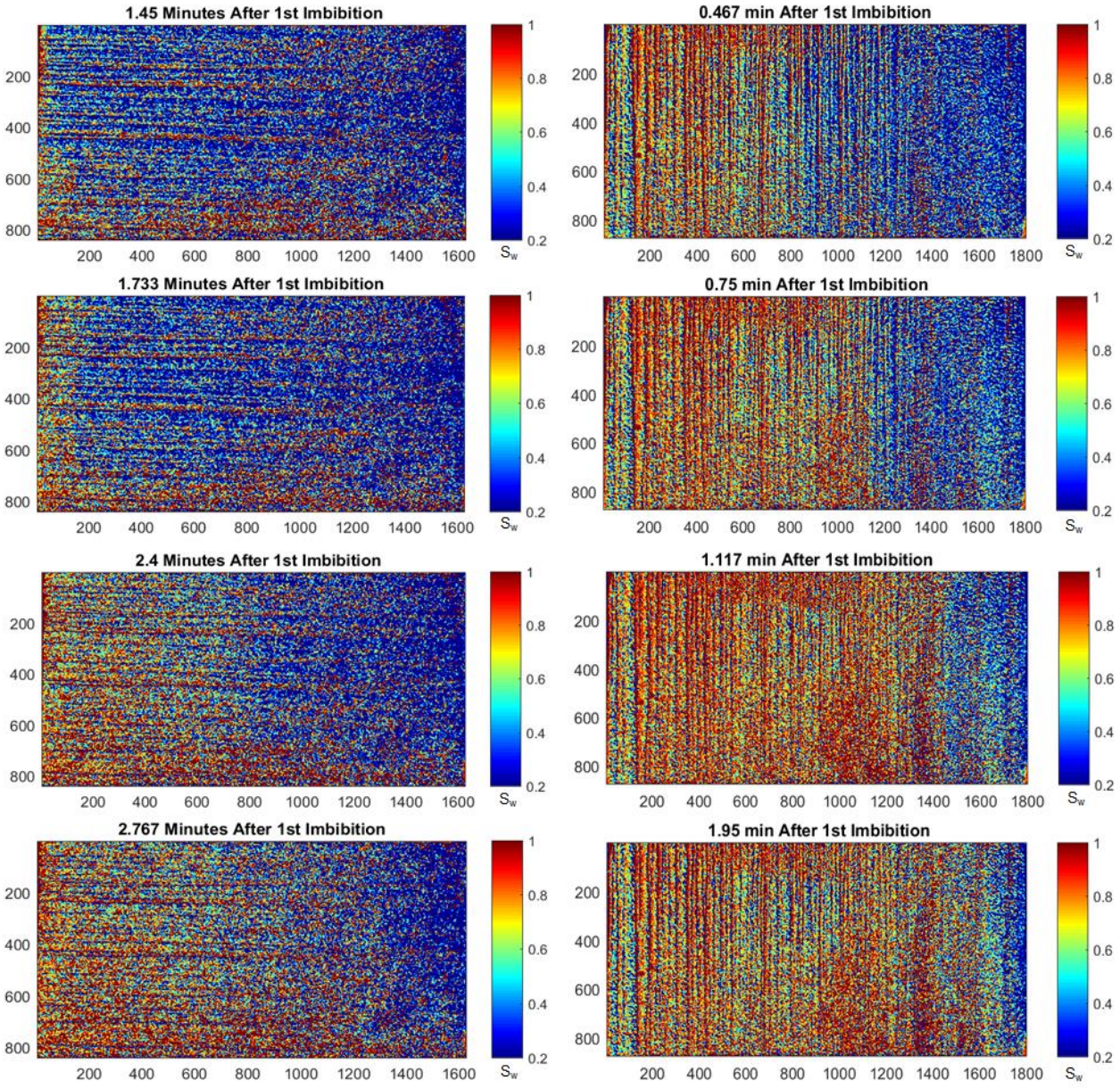

Fig. 1.42—Water saturation measurements of V2 vertical bedding (left) and H4 horizontal bedding (right) during primary imbibition.

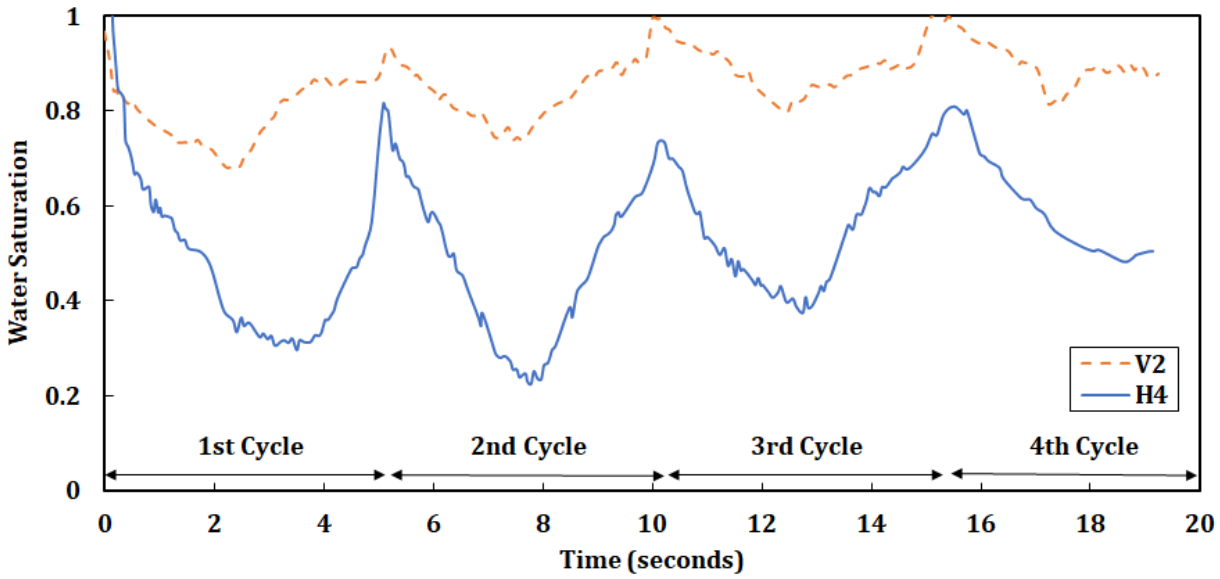

Fig. 1.43—Average water saturation of V2 vertical bedding and H4 horizontal bedding during 4 WAG cycles (larger water saturation in vertical bedding case is due to a more even displacement front).

1.4.3 Effect of Surfactant-Stabilized Foam

Fig. 1.44 shows the measured saturation map versus time of a horizontal bedding (H4) sample with surfactant stabilization during primary drainage. Like with previous horizontal bedding samples, fingering and early breakthrough are observed. The bedding orientation facilitates gas to flow through high permeability layers and leaves a large portion unswept during early time. **Fig. 1.45** shows the saturation map versus time of the same sample (H4) during primary imbibition but with 0.2% Bioterge AS-40 surfactant to stabilize the foam. This sample shows a laminated saturation pattern with an uneven displacement front. The primary imbibition process with surfactant achieves a smaller final water saturation after 3 minutes of brine injection. This could be explained by the increased pressure gradient required for flow once the surfactant solution is introduced.

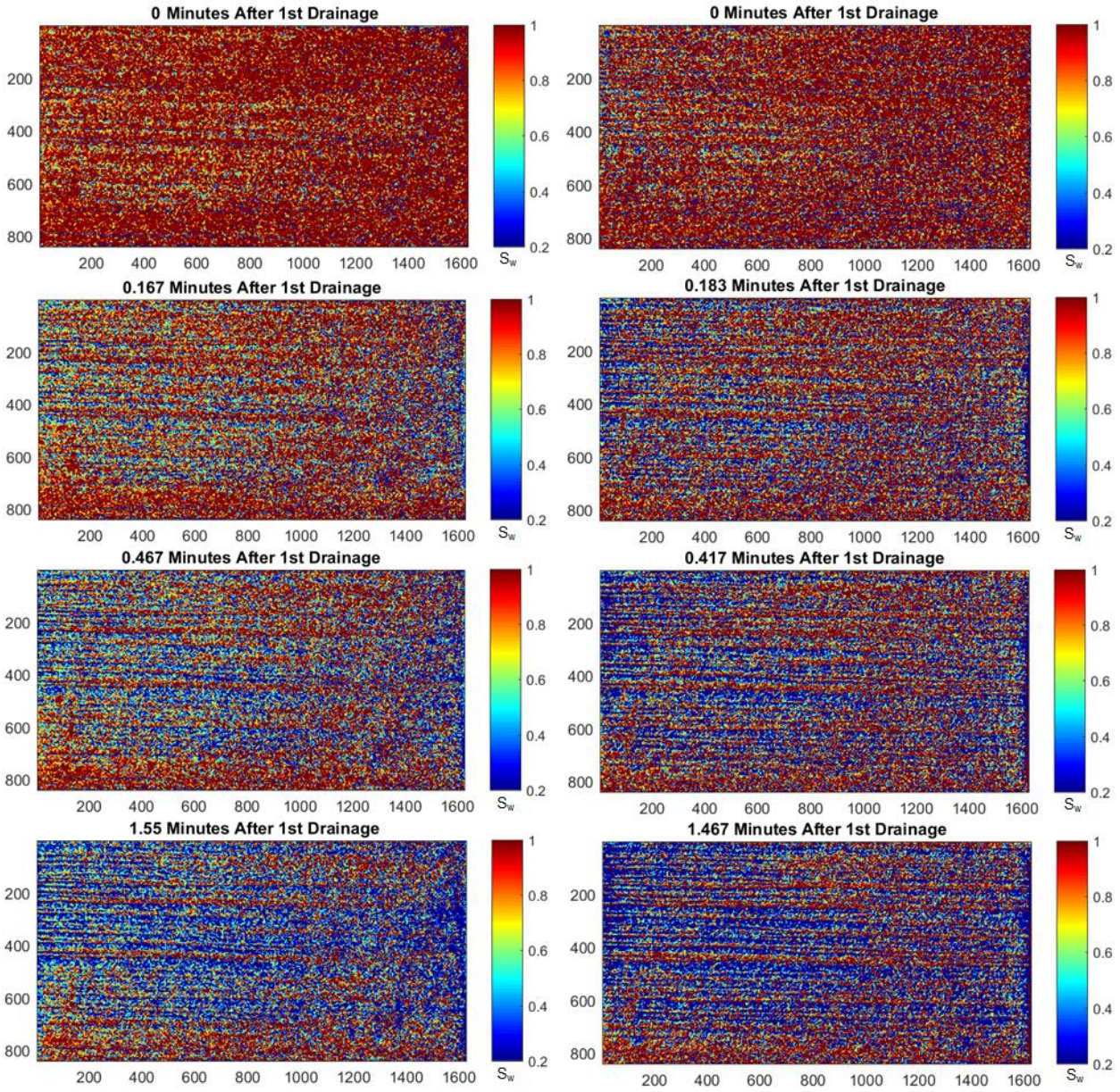

Fig. 1.44—Water saturation measurements of H4 horizontal bedding without surfactant (left) and H4 horizontal bedding with surfactant (right) during primary drainage.

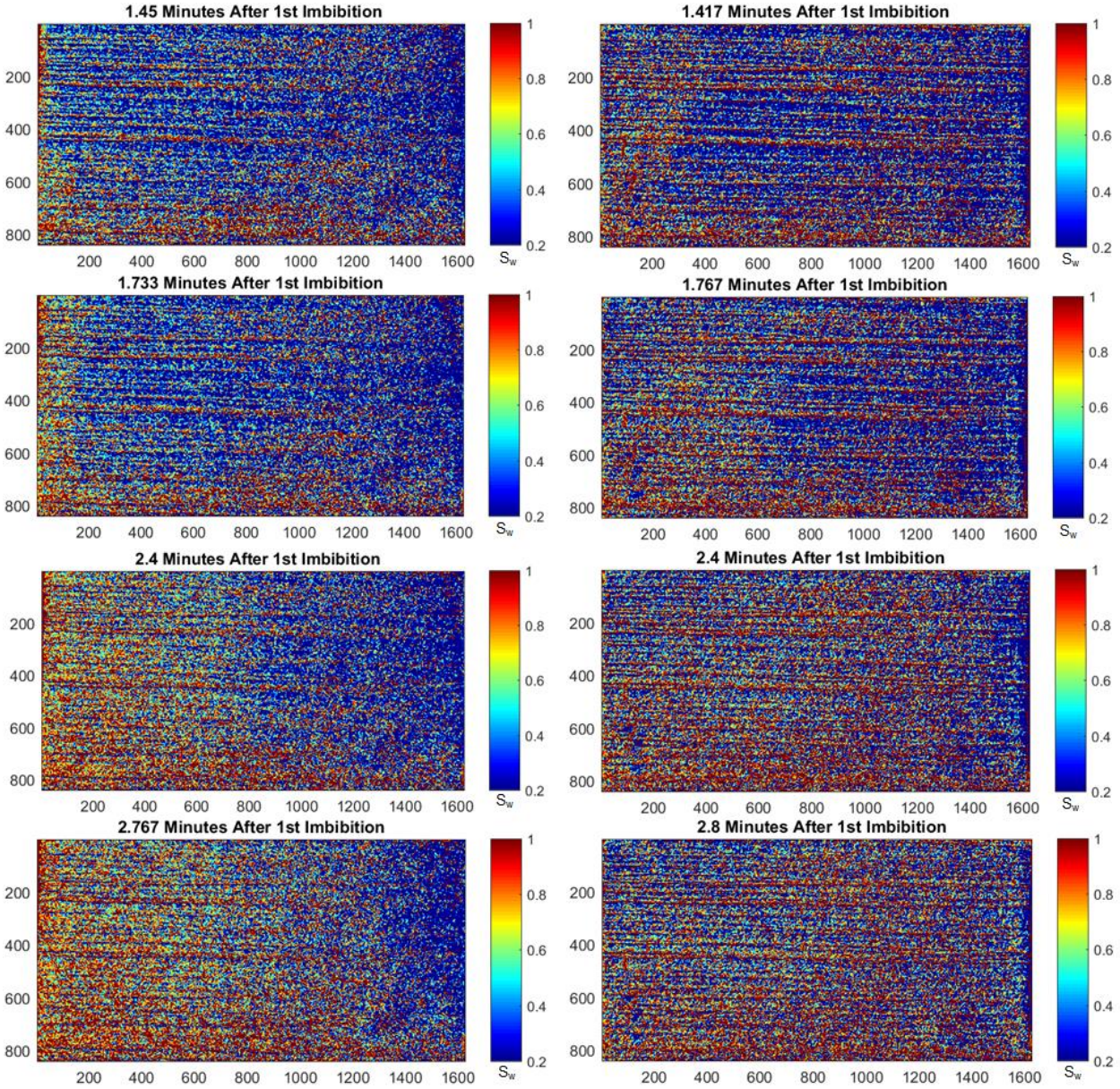

Fig. 1.45—Water saturation measurements of H4 without surfactant (left) and H4 with surfactant (right) during primary imbibition.

Fig. 1.46 shows the saturation map versus time of a horizontal bedding (H4) sample with surfactant stabilization during secondary drainage. Compared to previous experiments on the same horizontal bedding sample, the secondary drainage process with surfactant stabilization achieves more even and lower water saturations. Fingering is less prevalent and early breakthrough is reduced. This difference can be explained by a reduction in the mobility ratio of the gas phase from the surfactant-stabilized foam. This is confirmed with **Fig. 1.47**, which shows the average water saturation through 4 WAG cycles.

It is important to note that after the secondary imbibition of the horizontal bedding sample with surfactant stabilization, water saturation changes little with subsequent WAG cycling. We surmise that this is due to an increased minimum pressure gradient required for flow caused by the increased viscosity of foam. Since the injection-gas cylinder has already been partially depleted with previous drainage cycles, the remaining pressure within the gas cylinder is insufficient for mobilizing fluids in the rock sample. This parallels the field cases in which preformed foam has trouble propagating large distances due to the substantial minimum pressure gradient required for flow that develops at reservoir conditions. This hampers foam applications in far-wellbore environments. In these scenarios, foam is better utilized as a gas-blocking agent in the near-wellbore environment.

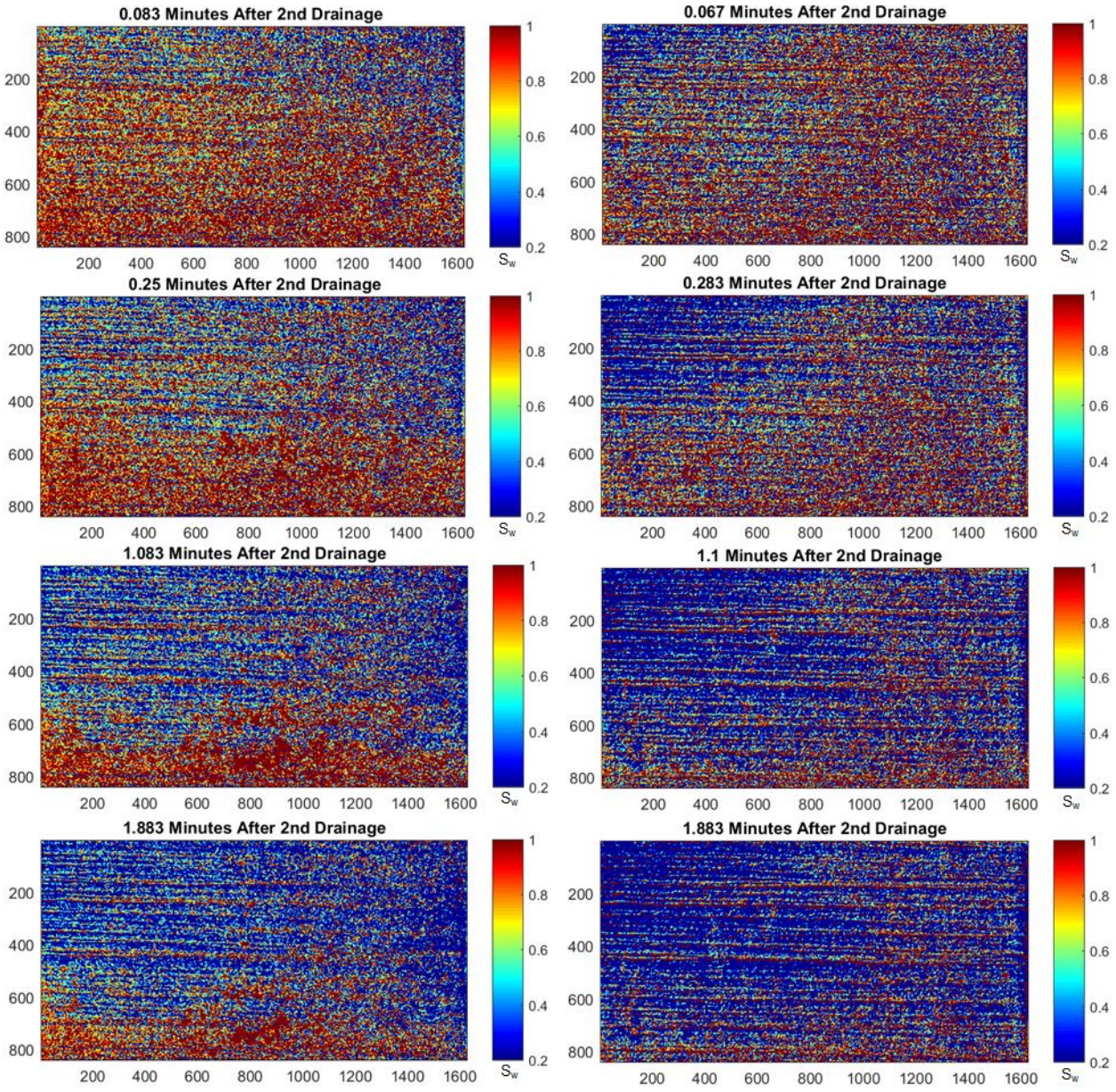

Fig. 1.46—Water saturation measurements of H4 without surfactant (left) and H4 with surfactant (right) during secondary drainage.

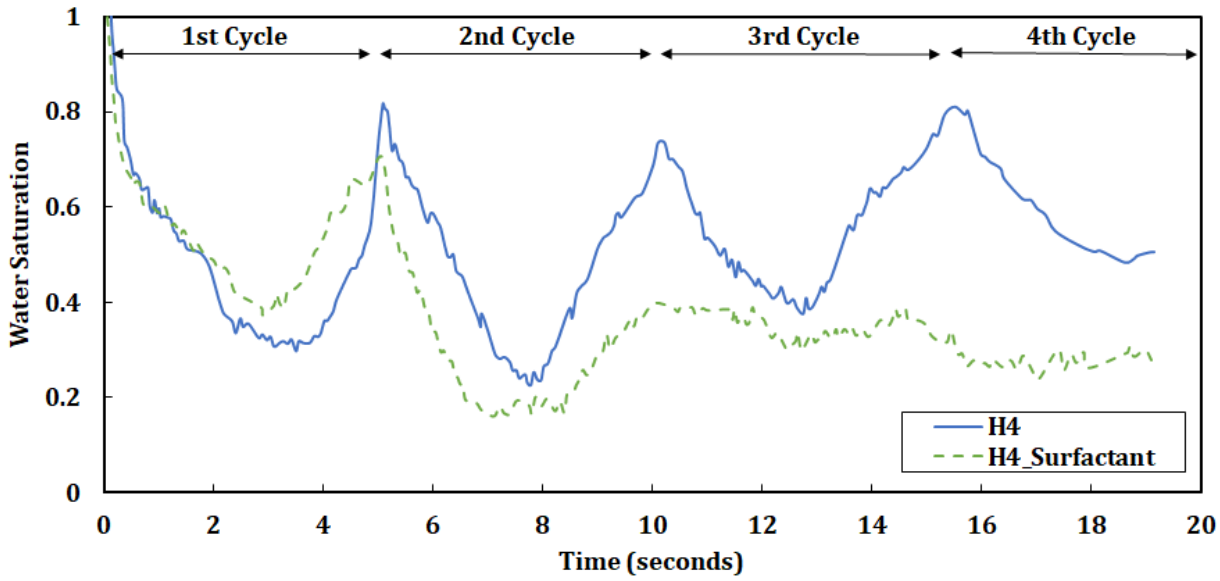

Fig. 1.47—Average water saturation images of H4 without surfactant and H4 with surfactant during 4 WAG cycles.

1.5 Numerical Simulations

Numerical simulations are conducted using CMG GEM in an attempt to match experimental results from CT scanning. Two cases are described in this section. The first with bedding direction perpendicular to flow. The second with bedding direction parallel to flow. A binary model consisting of two rock types (coarse and tight sandstone) are used for both cases in order to capture the rock heterogeneity of these laminated Berea sandstone samples.

1.5.1 Experimental Model Description

The model is first discretized into a grid block of dimensions 100 by 5 by 50. K-direction is taken to be down, meaning K = 50 corresponds to the bottom layer and K = 1 corresponds to the top layer. To scale the model to the actual sample size, block dimensions are calculated to be 0.00328084 ft in all directions. Recall, the samples have a length of 10 cm, height of 5 cm, and width of 0.5 cm. The model is binary since bedding is assumed to only consist of two rock types. Rock type 1 (RPT 1) corresponds to the coarse sandstone with a porosity of 17% and permeability of 100 mD. Rock type 2 (RPT 2) corresponds to the tight sandstone with a porosity of 5% and permeability of 5 mD in the I and J direction, and 1 mD in the K direction. Each grid block is assigned a rock type based on the downscaled binary model created from radiography images as described in Section 1.2.4.

1.5.2 Fluid Component Data

The Peng-Robinson equation is used for the fluid component model. A molar density of 1080 of kg/m^3 is assumed for the 10 wt% NaBr brine solution. Brine viscosity and isothermal compressibility are taken to be 0.70 cp and 3.3^{-6} psi^{-1}, respectively. Besides the aqueous phase of brine, the only other component considered is nitrogen. The relevant properties of nitrogen for input into GEM are described in **Table 1.3**.

Table 1.3—Properties of nitrogen gas.

Property	Value
Specific gravity	0.809
Average normal boiling point (°F)	-320.35
Molecular weight (gm/mol)	28.013
Acentric factor	0.04
Critical pressure (atm)	33.5
Critical volume (m^3/k-mol)	0.0895
Critical temperature (K)	126.2
Parachor	41

1.5.3 Rock Fluid Data

Two rock types are considered. Rock type 1 (RPT 1) corresponds to the coarse sandstone with a porosity of 17%. Rock type 2 (RPT 2) corresponds to the tight sandstone with a porosity of 5%. The below rock fluid properties are calculated using the Honarpour et al. correlation (1986):

- k_{rw}, the relative permeability to water at the given water saturation
- k_{row}, the relative permeability to oil at given water saturation.
- k_{rg}, the relative permeability to gas at the given saturation.
- k_{rog}, the relative permeability to oil in the presence of gas and connate water.

The relevant correlations are shown in **Eqs. 1.18** through **1.21** in which S_w is the water saturation, S_g is the gas saturation, S_o is the oil saturation, S_{wcon} is the endpoint saturation for connate water, S_{wcrit} is the endpoint saturation for critical water, S_{orw} is the endpoint saturation for residual oil of the water-oil table, S_{org} is the endpoint saturation for residual oil of the gas-liquid table, S_{gcrit} is the endpoint saturation for critical gas, and k_{rgcl} is the gas relative permeability at connate liquid.

$$k_{rw} = 0.0354\left(\frac{S_w - S_{wcrit}}{1 - S_{wcrit} - S_{orw}}\right) - 0.01087\left(\frac{S_w - S_{orw}}{1 - S_{wcrit} - S_{orw}}\right)^{2.9}$$

$$+ 0.566\, S_w{}^{3.6}(S_w - S_{wcrit}) \quad (1.18)$$

$$k_{row} = 0.76067\left(\frac{\frac{S_o}{1 - S_{wcon}} - S_{orw}}{1 - S_{orw}}\right)^{1.8}\left(\frac{S_o - S_{orw}}{1 - S_{wcon} - S_{orw}}\right)^{2.0}$$

$$+ 2.6318\emptyset(1 - S_{orw})(S_o - S_{orw}) \quad (1.19)$$

$$k_{rg} = 1.1072\, k_{rgcl}\left(\frac{S_g - S_{gcrit}}{1 - S_{wcon}}\right)^{2.0} + 2.7794\, S_{org}\, k_{rgcl}\left(\frac{S_g - S_{gcrit}}{1 - S_{wcon}}\right) \quad (1.20)$$

$$k_{rog} = 0.98372\left(\frac{S_o}{1 - S_{wcon}}\right)^{4.0}\left(\frac{S_o - S_{org}}{1 - S_{wcon} - S_{org}}\right)^{2.0} \quad (1.21)$$

The critical values needed for the correlations, including the endpoint saturations, are shown in **Table 1.4**.

Table 1.4— Values used in Honarpour et al. correlations.

Property		Sand	Shale
Endpoint Saturation: Connate Water	S_{wcon}	0.2	0.25
Endpoint Saturation: Critical Water	S_{wcrit}	0.2	0.25
Endpoint Saturation: Residual Oil for Water-Oil Table	S_{orw}	0.2	0.25
Endpoint Saturation: Residual Oil for Gas-Liquid Table	S_{org}	0.2	0.25
Endpoint Saturation: Critical Gas	S_{gcrit}	0.2	0.25
Relative Gas Permeability at Connate Liquid	k_{rgcl}	1	1

Using these values, the water-oil and gas-liquid tables are calculated for both rock types as shown in **Tables 1.5** and 1.**6**, and **Figs. 1.48** and **1.49**.

Table 1.5—Water-oil and gas-liquid table for rock type 1 (coarse sandstone).

SWT			SGT		
S_w	k_{rw}	k_{row}	S_g	k_{rg}	k_{rog}
0.20	0	0.975	0	0	0.975
0.24	0.002	0.801	0.1	0	0.397
0.28	0.005	0.653	0.2	0	0.137
0.31	0.008	0.529	0.225	0.018	0.102
0.35	0.011	0.426	0.25	0.039	0.0742
0.39	0.014	0.340	0.275	0.062	0.0531
0.43	0.018	0.270	0.3	0.087	0.0372
0.46	0.024	0.214	0.325	0.114	0.0255
0.50	0.030	0.168	0.35	0.143	0.0170
0.54	0.038	0.132	0.375	0.175	0.0109
0.58	0.048	0.102	0.4	0.208	0.00677
0.61	0.061	0.079	0.425	0.244	0.00401
0.65	0.076	0.059	0.45	0.282	0.00223
0.69	0.094	0.042	0.475	0.322	0.00115
0.73	0.117	0.027	0.5	0.364	0.000536
0.76	0.144	0.013	0.525	0.409	0.000213
0.80	0.176	0	0.55	0.455	0.0000646
0.90	0.295	0	0.575	0.504	0.0000106
1.00	0.475	0	0.6	0.555	0
0.20	0	0.975			
0.24	0.002	0.801			

Table 1.6—Water-oil and gas-liquid table for rock type 2 (tight sandstone).

SWT			SGT		
S_w	k_{rw}	k_{row}	S_g	k_{rg}	k_{rog}
0.25	0	0.790	0	0	0.790
0.28	0.002	0.631	0.13	0	0.214
0.31	0.005	0.497	0.25	0	0.0390
0.34	0.008	0.386	0.27	0.015	0.0302
0.38	0.011	0.294	0.28	0.031	0.0231
0.41	0.014	0.221	0.30	0.048	0.0174
0.44	0.018	0.162	0.31	0.066	0.0129
0.47	0.023	0.116	0.33	0.084	0.00935
0.50	0.028	0.081	0.34	0.104	0.00664
0.53	0.034	0.055	0.36	0.125	0.00460
0.56	0.042	0.036	0.38	0.147	0.00309
0.59	0.050	0.023	0.39	0.169	0.00199
0.63	0.061	0.014	0.41	0.193	0.00123
0.66	0.073	0.008	0.42	0.217	0.000707
0.69	0.088	0.004	0.44	0.243	0.000372
0.72	0.105	0.002	0.45	0.269	0.000171
0.75	0.125	0	0.47	0.297	0.0000610
0.88	0.242	0	0.48	0.325	0.0000121
1	0.442	0	0.5	0.355	0

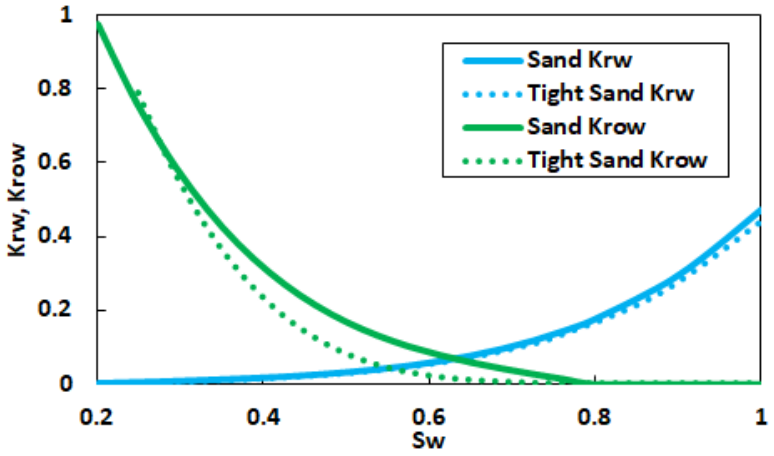

Fig. 1.48—Water-oil relative permeability graph.

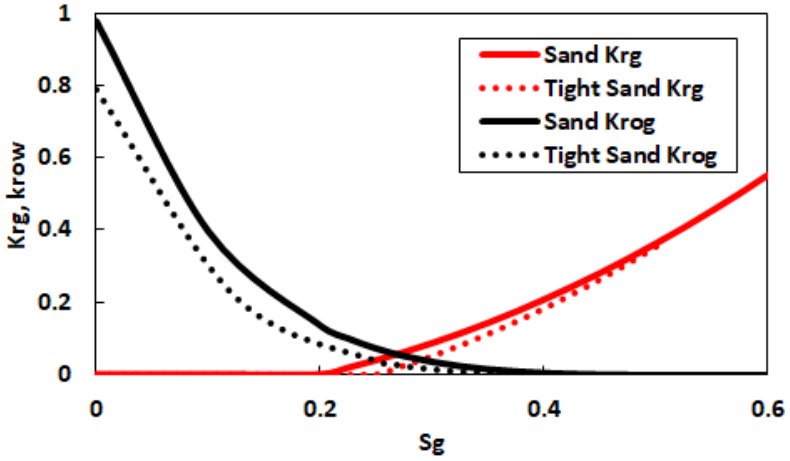

Fig. 1.49—Gas-liquid relative permeability graph.

1.5.4 Well Data

For the simulator to calculate the well index, well geometric characteristics are specified first. The wellbore is set parallel to the I axis – the same direction in which gas is injected. A wellbore radius of 0.000164 feet, geometric factor of 0.37, and a skin factor of 0 are assumed. Production wells' surface water rate are capped at 1 cc/min, which is the total injection rate. Perforations are directed parallel to the I coordinate axis.

1.5.5 Results

1.5.5a Vertical Bedding Direction

Figs. 1.50 and **1.51** show the corresponding CMG water saturation prediction versus experimental results after primary gas injection for approximately 3.3 minutes. The bedding of the slab is perpendicular to the flow direction. The strip-wise saturation pattern is a result of the bedding orientation. In this case, no fingering is observed. Furthermore, the displacement front is relatively uniform and sweep efficiency is high.

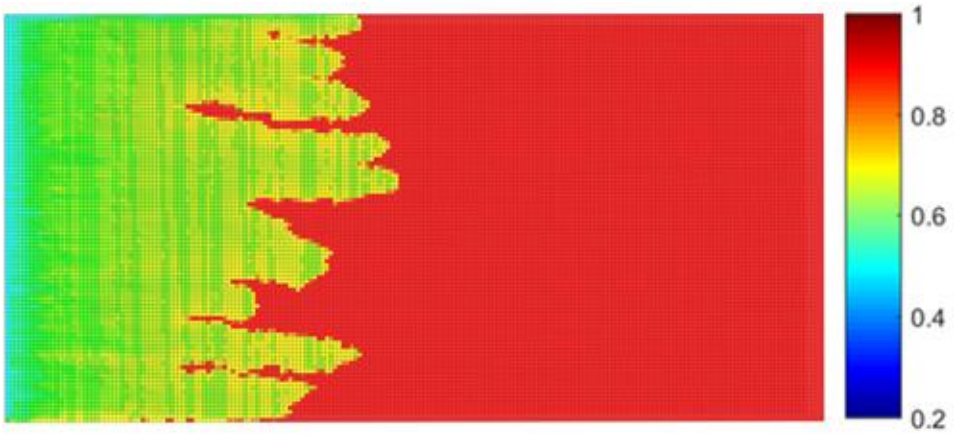

Fig. 1.50—Water saturation simulation image during primary gas injection (from left to right).

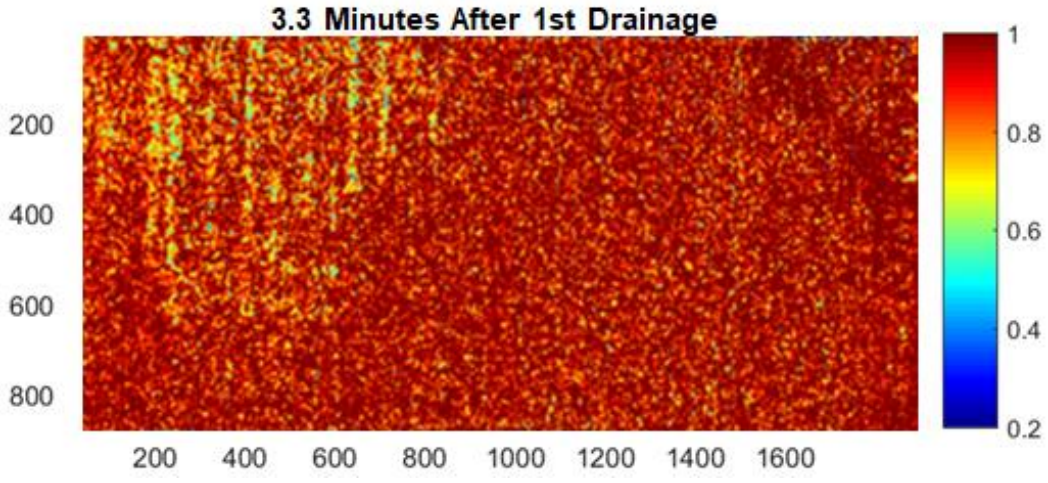

Fig. 1.51—Water saturation images during primary gas injection (from left to right).

1.5.5b Horizontal Bedding Direction

Figs. 1.52 and **1.53** show the corresponding CMG water saturation prediction versus experimental results after primary gas injection for approximately 1.1 minutes. The bedding of the slab is parallel to the flow direction. The laminated saturation pattern is a result of the bedding orientation. Fingering and early breakthrough are observed. This bedding orientation facilitates gas to flow through high permeability layers and leaves a large portion unswept at the beginning. After tens of PV of gas are injected, the rock sample is evenly reduced to lower water saturations.

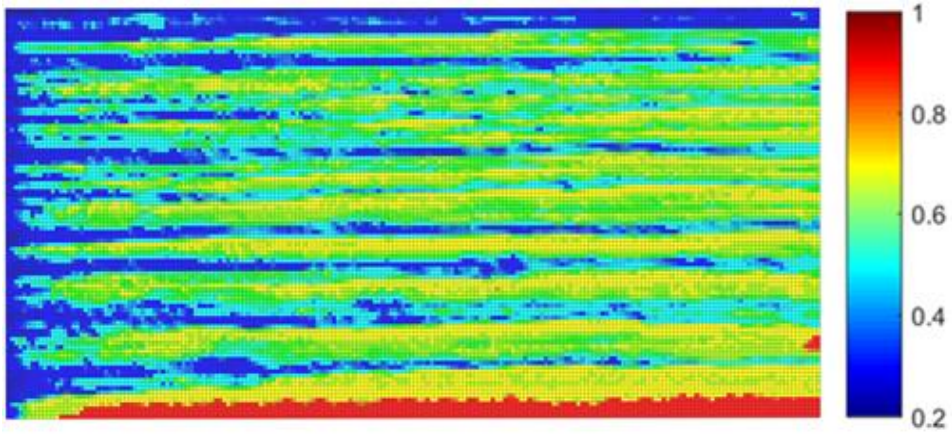

Fig. 1.52—Water saturation simulation image during primary gas injection (from left to right).

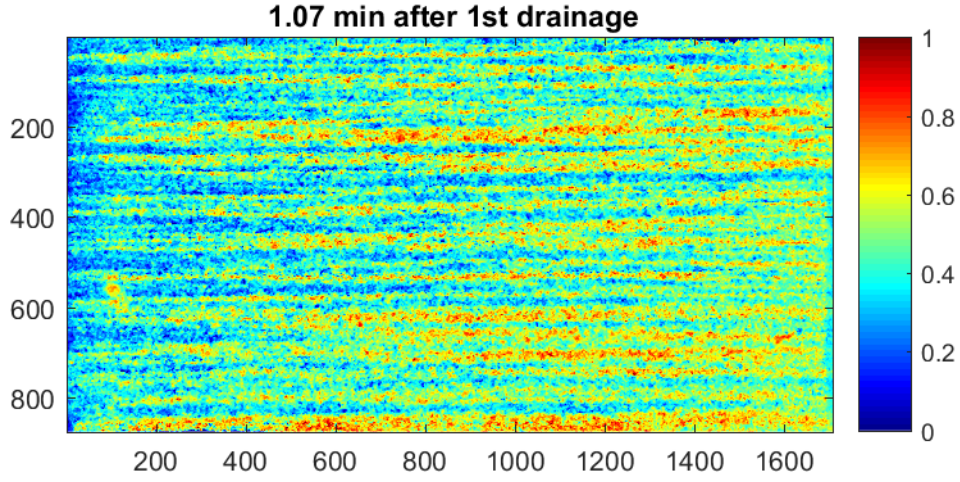

Fig. 1.53—Water saturation CT image during primary gas injection in type-2 rock (from left to right).

Fig. 1.54 shows the numerical water saturation predictions versus time for a horizontal bedding sample (H4) and vertical bedding sample (V2) during primary drainage. Similar to the experimental results, fingering occurs with the horizontal bedding sample and records an early breakthrough time of 0.46 min. The laminated saturation pattern is clearly visible. The numerical water saturation predictions for the vertical bedding sample, however, are less comparable to the experimental results. While fingering is reduced and the displacement front is more even, the overall saturation pattern looks quite different. We presume the difference is owed to sharp thresholding assumptions between the coarse sandstone rock type and the tight sandstone rock type, instead of a gradual change of transport properties occurring in the actual rock sample.

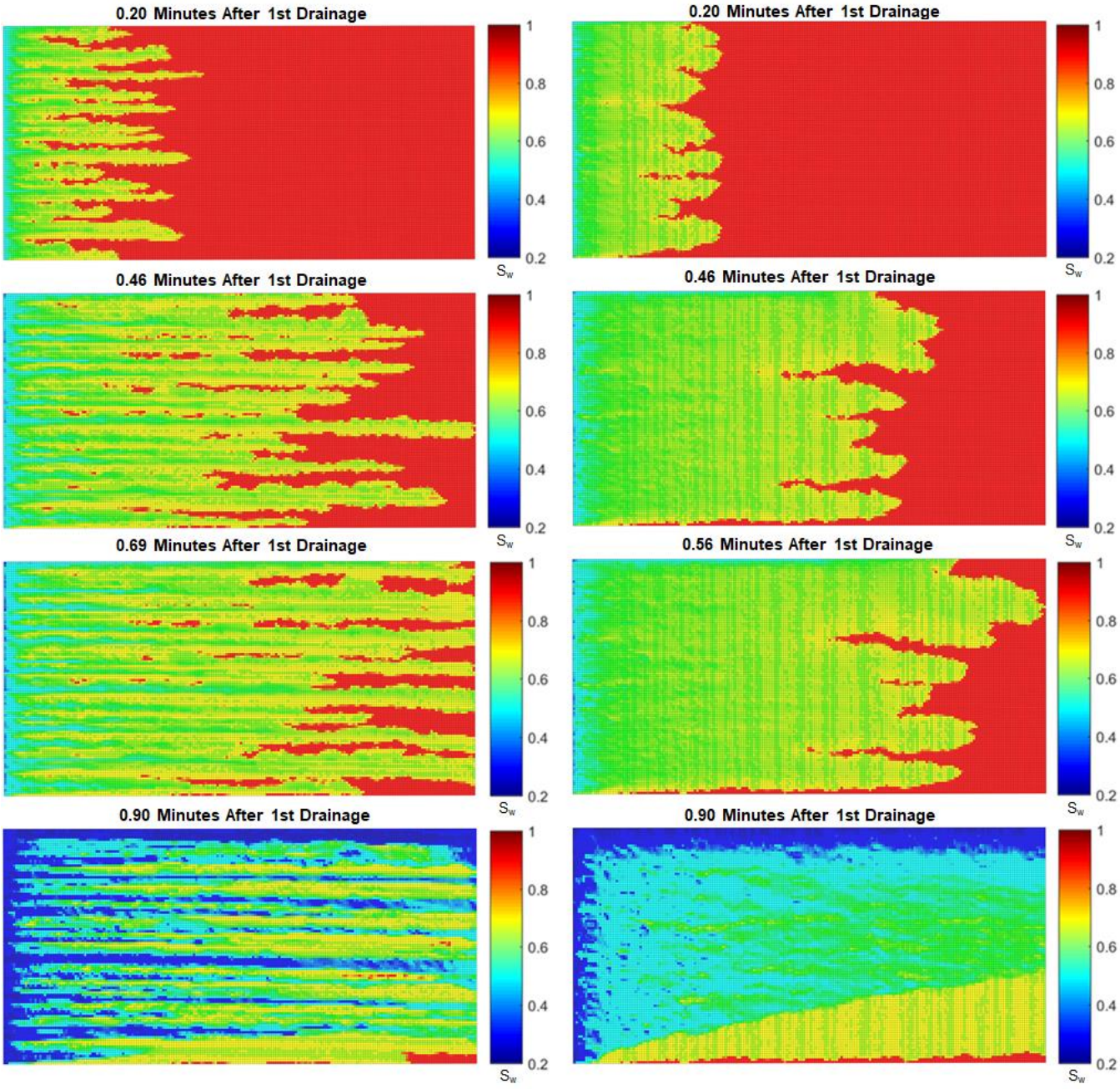

Fig. 1.54—Water saturation predictions of H4 horizontal bedding (left) and V2 vertical bedding (right) during primary drainage.

1.6 Conclusions

In this study, we utilize an automated fluid injection system monitored by an X-ray micro-focus scanner to quantify displacement patterns and saturations during WAG core-flood and mobility control experiments. Based on our results, the following conclusions are drawn:

- Successful design of an experimental apparatus that can image flow processes within a micro-CT machine permitted real-time monitoring of multiphase fluid flow processes with a time resolution of 500 ms and a spatial resolution of 60.6 μm.
- As expected, saturation pattern and displacement front characteristics during WAG injection are highly influenced by bedding orientation and rock heterogeneity. Our results show a difference of 18% – 37% in average water saturation between vertical bedding and horizontal bedding samples across 4 WAG cycles.
- Without surfactant stabilized foam during WAG injection, channeling and early breakthrough occur in those cases in which bedding orientation facilitates gas to flow through high permeability layers. With 0.2% Bioterge AS-40 surfactant, average water saturation was decreased by as high as 25% during the secondary drainage cycle.
- Foam promotes larger mobility reductions in high permeability zones as compared to lower permeability zones with differences in water saturation as large as 27% and thus improves overall sweep efficiency.

Chapter 2: 3D Mapping of Fracture Networks in Shale Cores

Jeffery S. Luo, Matthew J. Ramos, Lawrence Chen, and D. Nicolas Espinoza

2.1 Introduction

Understanding the mechanical and physical properties of shale rock is becomingly increasingly important with the recent shift to unconventional production of hydrocarbon resources in the US (EIA 2018). Since shale porosity and permeability is very low, production from these reservoirs is only economically viable with recent technological advances in hydraulic fracturing and horizontal drilling. The efficacy of these techniques is heavily influenced by subsurface heterogeneity, including pre-existing fractures and lithological differences that can cause anisotropic mechanical properties (Bodziak et al. 2014, Gale et al. 2017, Lee et al. 2015, Ramos 2018). For instance, heterogeneities and rock laminations aligned in a certain orientation can act as planes of weakness, thereby facilitating the growth of induced hydraulic fractures (Ramos 2018). Conversely, these features can also limit the growth of hydraulic fractures and increase wellbore stabilities issues (Mokhtari et al. 2014).

To understand the potential for pre-existing fractures and rock heterogeneities to aid or hinder the propagation of fractures induced, a detailed characterization of these features is needed. The relative size and orientation of these fractures need to be analyzed in relation to mechanical rock properties. Much of this work is a continuation from work done by Matt Ramos and Lawrence Chen. Ramos (2018) quantified the impacts of layering and fracturing on shale mechanical properties by conducting quasi-static strain measurements and ultrasonic P- and S- wave propagation under various stress loading paths. Lawrence Chen began visualizing some of these centimeter core plugs using Fiji/Image in an attempt to understand how rock layering and pre-existing fractures influence the propagation of fractures induced from tri-axial loading.

2.2 Materials and Methods

2.2.1 Rock Types

Mancos shale core plug samples are used for this study. Dimensions of core plugs were 50 mm x 25 mm, a 2 to 1 ratio of length to diameter. End faces were grounded to ensure parallelism. Tests were conducted on air-dried samples, so pore pressure was assumed to be negligible. Mancos Shale is a gas producing source rock within the Uinta Basin, which resides in parts of eastern Utah, Colorado, and Wyoming. As shown in **Fig. 2.1**, Mancos samples exhibit two major facies in their layering. Mancos shale samples were taken at various orientations with respect to bedding (0°, 45°, and 90°). Individual layers commonly deviated from these orientations due to the rock heterogeneity and cross-bedding. Scanning Electron Microscope (SEM), Back-Scattered Electron Detection (BSE) and Energy Dispersive X-ray Spectroscopy (EDS) measurements show that the lighter facie consists of roughly 52% quartz, 13% clays, 16% calcite, and 11.4% dolomite, whereas the darker facie consists of roughly 15% quartz, 46% clays, 5% calcite, and 19% dolomite with minor minerals making up the remainder (Ramos 2018).

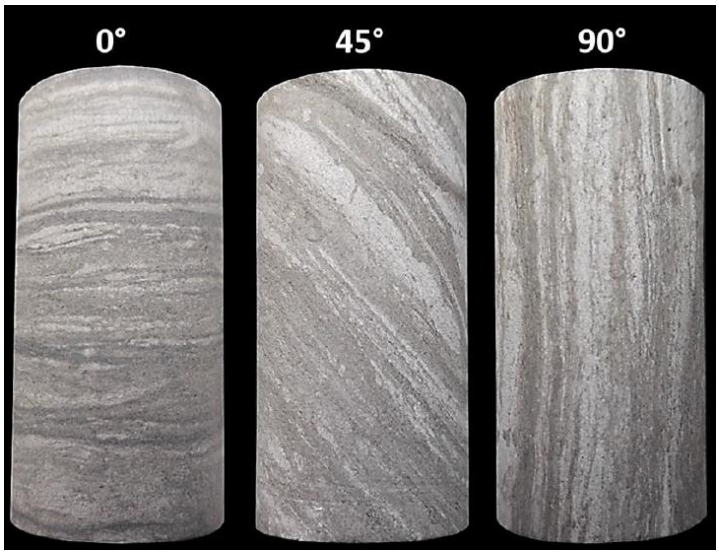

Fig. 2.1—Modified figure from Ramos (2018) showing Mancos core samples with bedding at 0° (left), 45° (middle), and 90° (right).

2.2.2 Triaxial Load Frame and Testing Conditions

Loading to failure was conducted using a Terratek triaxial load frame capable of applying axial loads up to 225,000 kg and total radial stresses up to 138 MPa. Axial and radial displacements were calculated from readings provided by cantilever arms. Axial displacements are an average of four cantilever arms. Radial displacements are an average of two perpendicular radial displacements each measured from two cantilever arms (Ramos 2018). Strain gauges were calibrated with a Mitutoyo micrometer and have a precision of $\pm 0.5 \cdot 10^{-7}$ (Ramos 2018).

Stresses and strains are measured in the principal direction with the confining stress radially symmetric (**Fig. 2.2**). We refer to σ_1 and σ_3 as the maximum and minimum principal stresses. Hence, the relative orientations of stresses and strains are defined accordingly to the axial (σ_1, ε_1) or radial (σ_3, ε_3) directions.

Fig. 2.2—Pictures of the Terratek triaxial load frame (left), triaxial cell (middle), and the corresponding schematic of the triaxial cell (right).

All samples were loaded axially at a strain rate of $\dot{\varepsilon}_1 = 4 \cdot 10^{-6}\, s^{-1}$ with constant radial confining stress until failure. Tests were performed at differing radial stresses to evaluate the impact of confining stress on damage evolution, stiffness anisotropy, and nonlinearity in Mancos samples. Interested readers should refer to Matt Ramo's Ph.D. dissertation (2018) for details as those results will not be covered here.

2.2.3 X-Ray System

X-ray micro-computed tomography (CT) imaging is a nondestructive imaging technique that gives 3D images of the solid interiors based on X-ray attenuation. CT imaging is performed with a Nikon XTH-225 scanner and CT dataset reconstruction is done using CTPro3D, a software developed by Nikon. The default 225 kV X-ray source is equipped with a reflection target offering a 3 μm spot size. Mancos samples were scanned both before testing and after triaxial testing to failure. Pre-scanning was used to identify any defects or pre-existing fractures and determine the rock's microstructure. Post-testing imaging was used to evaluate the effect of existing bedding planes and pre-existing fractures on stress-induced fracture development. The potential impact of these factors on the orientation, branching, and containment of stress-induced fractures during testing has been studied before (Germanovich and Astakhov 2004, Suarez-Rivera et al. 2013). Ramos (2018) was able to extend these studies to different rock samples like Mancos shale. This study continues some of the work done by Ramos.

2.2.4 Image Processing and Analysis

After a CT scan is reconstructed, a stack of image files are outputted. The image files are in TIF format. Each pixel of a single image has a grayscale value ranging from 0 to 65535 which is linearly dependent on material density and X-ray power. The stack of images are fed through Fiji/Image-J in order to interpret and map the fracture network. The procedure for this workflow is summarized in **Fig. 2.3**.

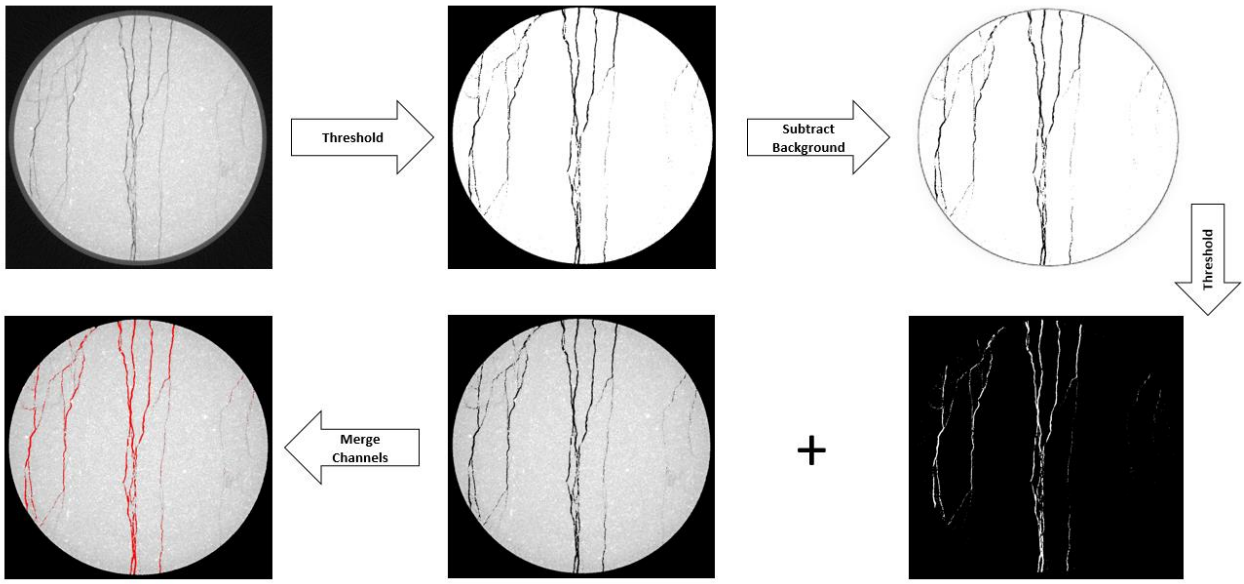

Fig. 2.3—Image analysis workflow using Fiji/Image-J to map fracture network.

First, the image files are imported as a stack (File → Import → Image Sequence). It is usually necessary to remove the end slices of the stack since these images do not include any relevant information (Image → Stacks → Tools → Slice Remover). Brightness and contrast are adjusted to accentuate the fracture features and the background rock (Image → Adjust → Brightness/Contrast → Auto). These steps are outlined in **Fig. 2.4** and the result is shown with **Fig. 2.5**.

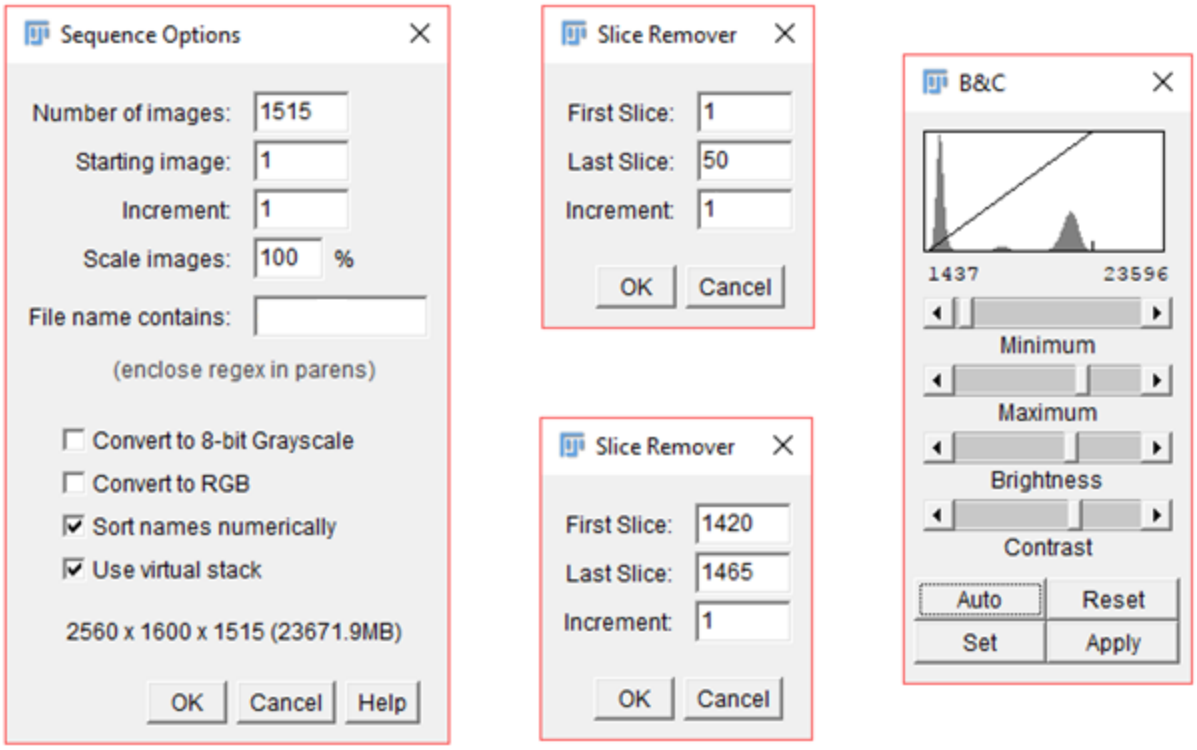

Fig. 2.4—Stack of images are read (left), unnecessary slices removed (middle), and contrast adjusted (right).

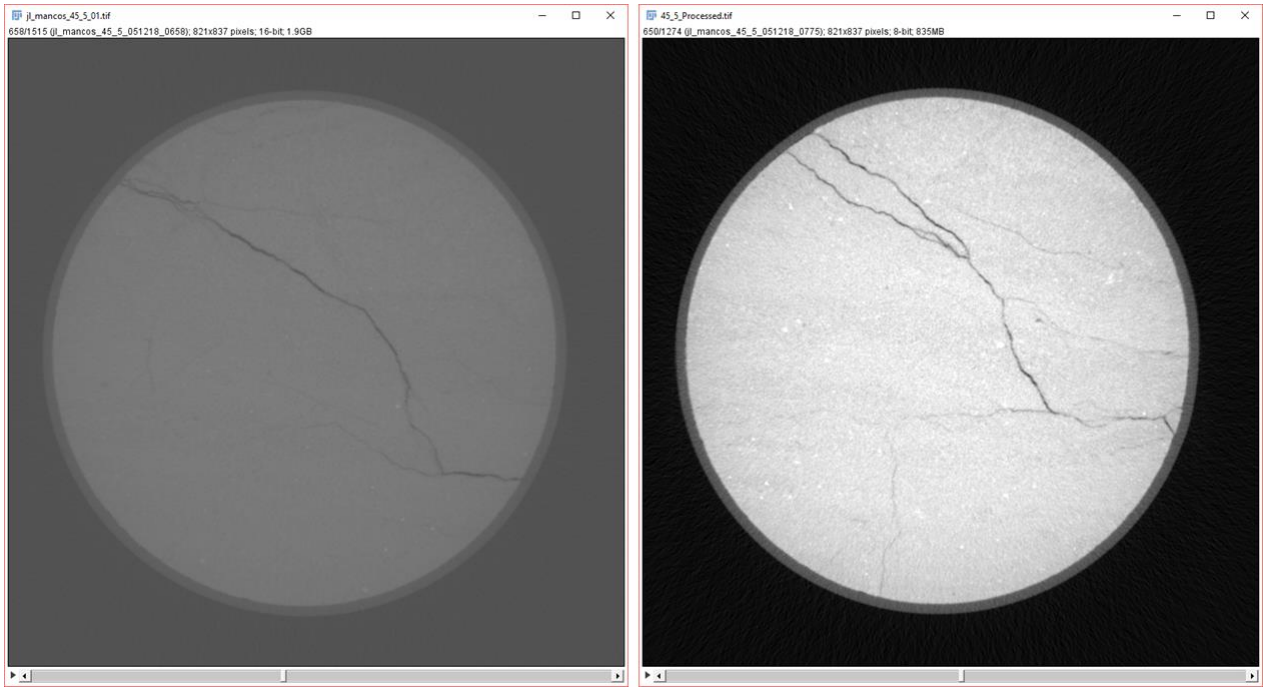

Fig. 2.5—Original stack of images (left) and stack after contrast is adjusted and end slices removed.

Image is converted from 16-bit to 8-bit since subsequent steps require an 8-bit image (Image → Type). After, a threshold is applied to the processed images in order to segment the fracture from the background rock (Image → Adjust → Threshold). Segmentation is necessary in order to isolate and analyze the fracture network. Fiji/ImageJ has built-in algorithms for the segmentation process. The default segmentation options are of the simple, global thresholding variant that works well with high quality images. The segmentation step is outlined in **Fig. 2.6** and the result is shown with **Fig. 2.7**. Too high of a threshold limit results in slightly better fracture segmentation but at the cost of more noise localized around the edges of the sample. This is due to imaging artefacts caused by blurring and possible overcorrection of beam-hardening effects. Further discussion of the segmentation process is documented in Section 2.4.

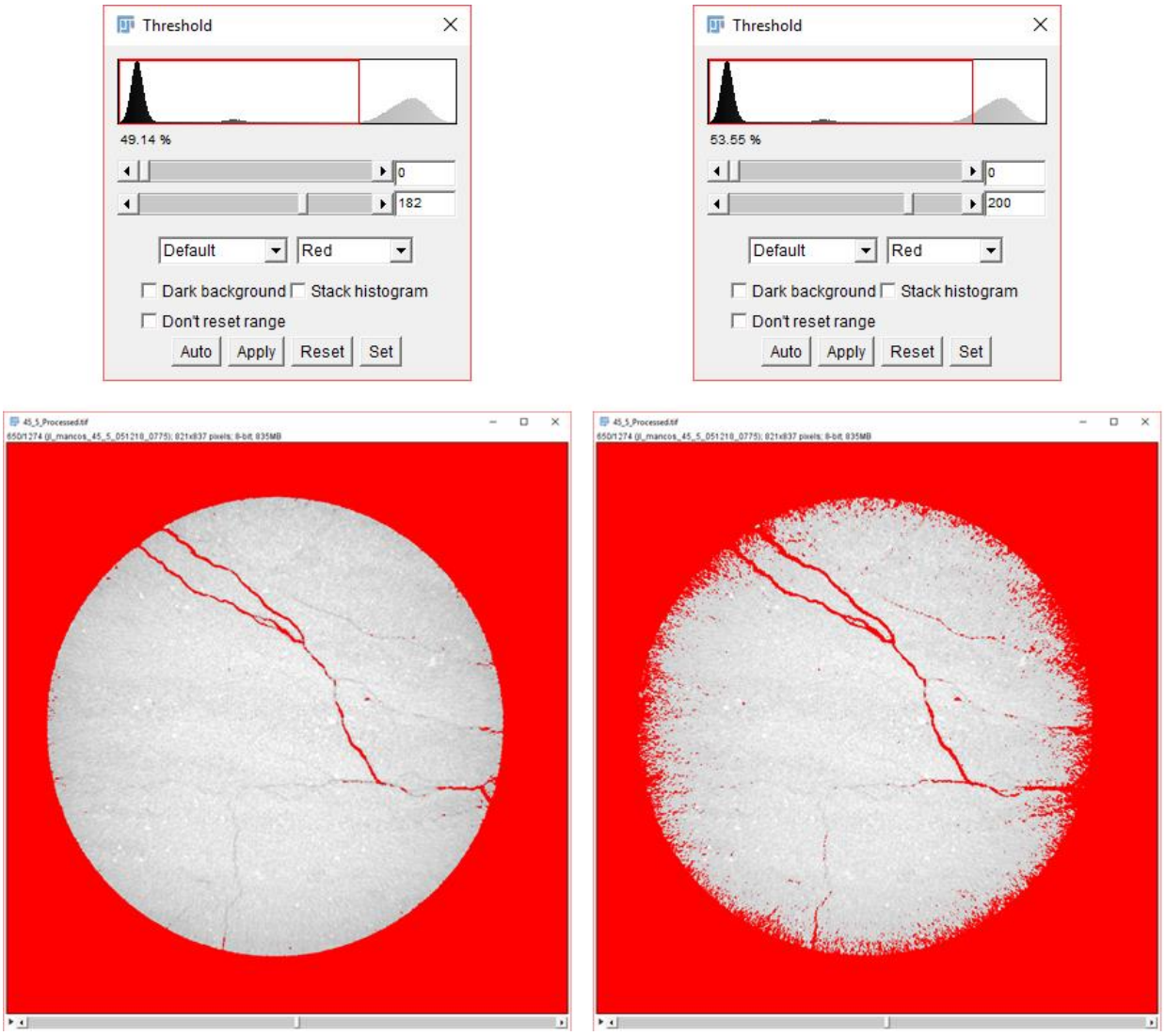

Fig. 2.6—Segmentation results from two different threshold limits. The greater the limit the more noise.

Images that are very noisy may need post-processing such as filtering before segmentation is attempted. No filtering was done in this study since the original images were already of suitable quality. While further post-processing and convolution operations could possibly improve the segmentation process it is at the cost of information blurring and loss. In this study, little improvement in segmentation was derived from filtering techniques like Gaussian blur, mean, and median.

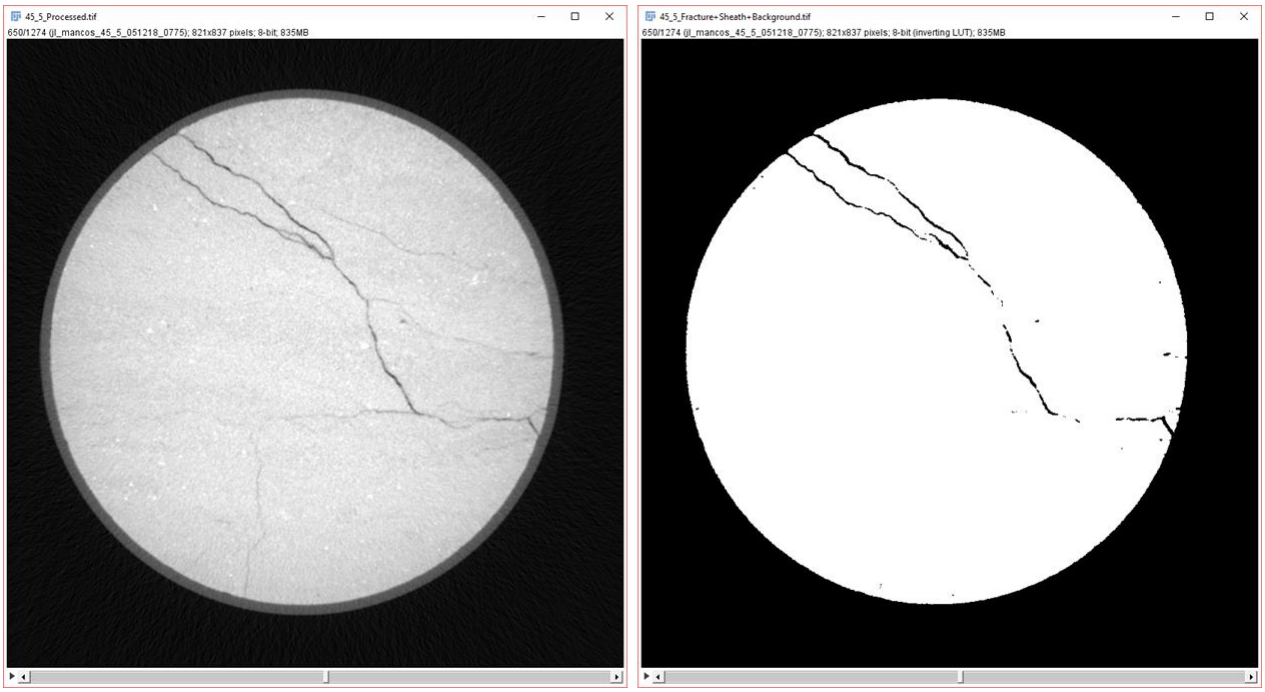

Fig. 2.7—Processed stack of images (left) and stack after segmentation.

The background is subtracted (Process → Subtract Background) and another threshold is applied to isolate the fracture network and remove irrelevant features such as the plastic sheath wrapped around the rock, which was used to maintain the integrity of the sample during triaxial testing. These steps are outlined in **Fig. 2.8** and the result is shown with **Fig. 2.9**.

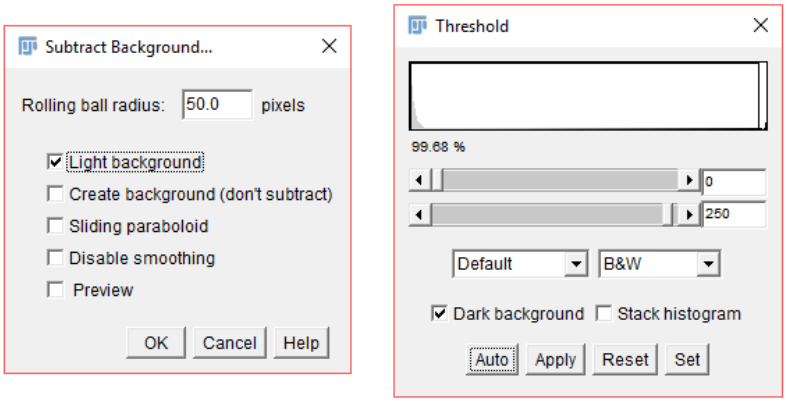

Fig. 2.8—Background is subtracted (left) and another threshold is applied (right).

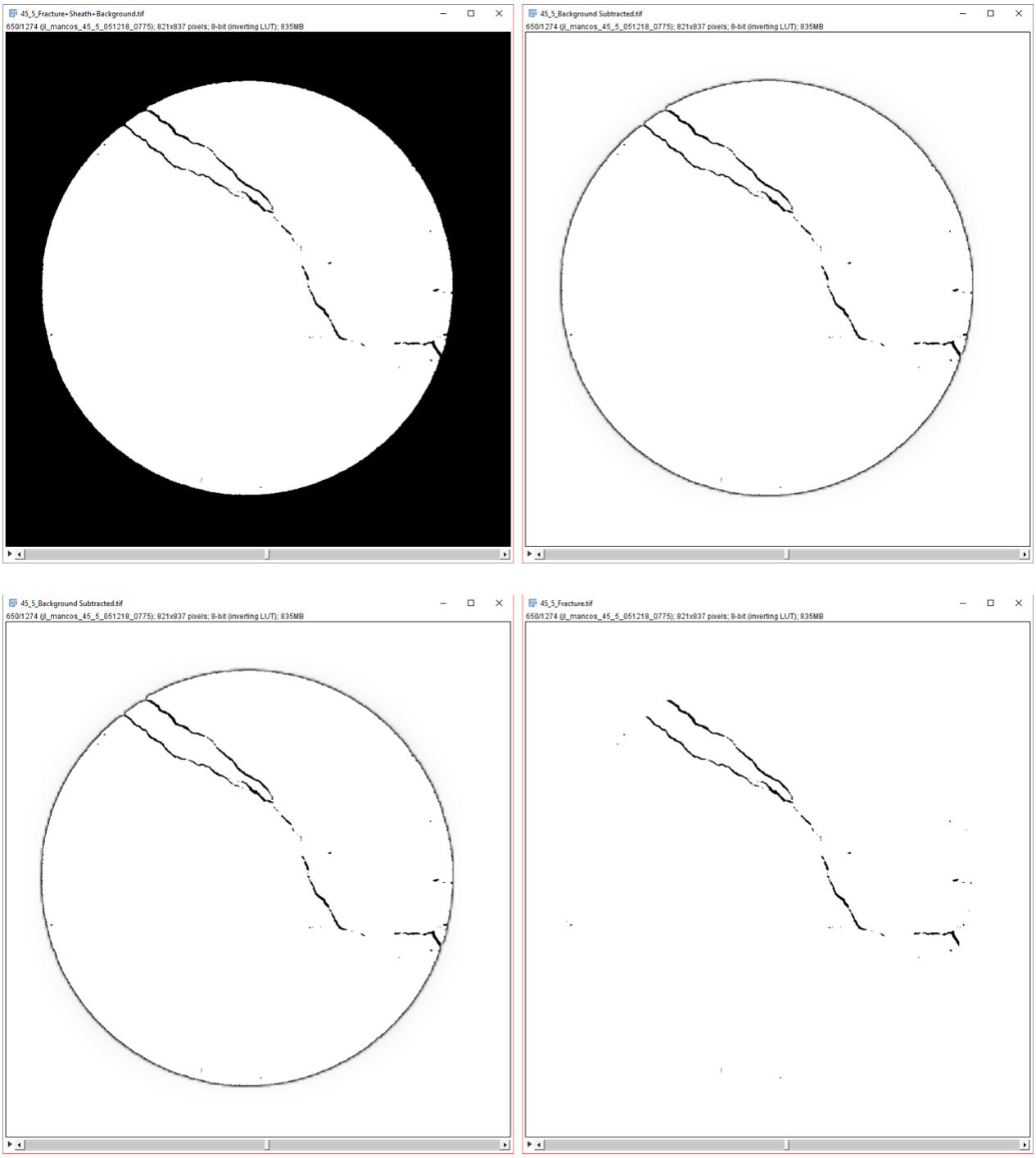

Fig. 2.9—Segmented stack (top left), stack after subtracting background (top right), and stack after another threshold is applied to remove the outer ring (bottom right).

In order for the isolated fracture to be viewed correctly in 3D (Plugins → 3D Viewer), the stack should be first inverted (Edit → Invert). Black and white values are inverted with this step (i.e. fracture in black has a value of 255, but after inversion the fracture is now white and has a value of 0). Now, in 3D viewer a lower threshold of 50 or anything larger than 0 can be set in order to correctly render the fracture volume, which has an isovalue of 0 after inversion. These steps are outlined in **Fig. 2.10** and **Fig. 2.11**.

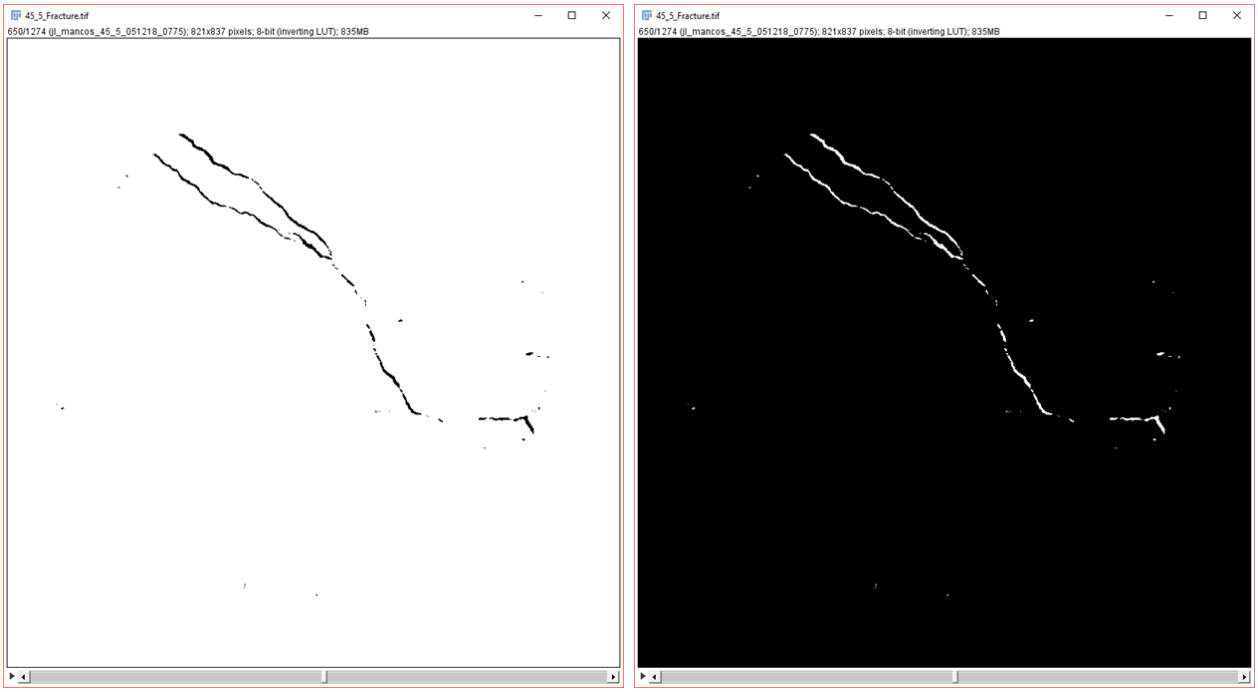

Fig. 2.10—Isolated fracture stack (left) and stack after inversion (right).

In the 3D viewer window, the user can specify how to display the fracture. For the case shown in **Fig. 2.11**, a volume is displayed and rendered as red with a resampling factor of 4. The user can choose several displays (Add → From Image → Display as) including volume, orthoslice, surface, etc. Increasing the resampling factor from a default of 2 leads to quicker rendering time as the images are downsampled for display.

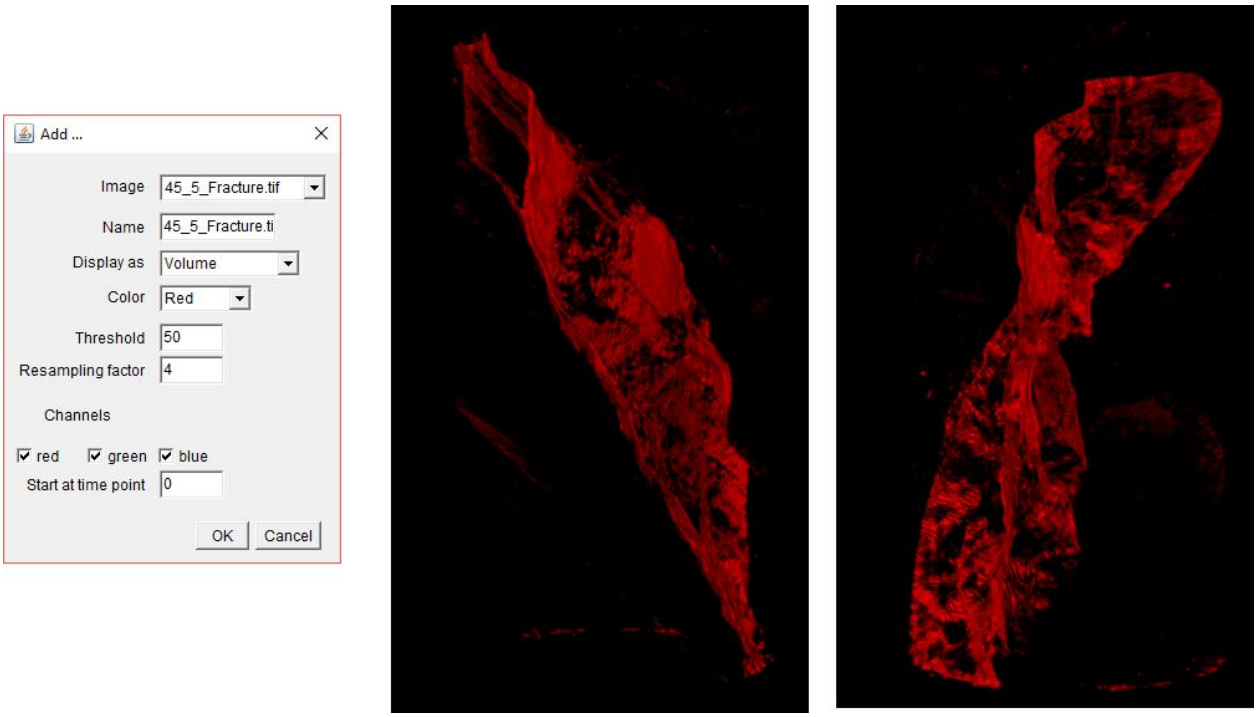

Fig. 2.11—3D viewer settings for fracture visualization (left) and its corresponding rendering (right).

The segmented fracture can also be superimposed onto the rock background by using the 'Image Calculator' (Process → Image Calculator). By taking the minimum of the processed image and the inverted segmented image, one can generate a superimposed fracture outline onto the rock background (**Fig 2.12**). The inverted segmented image shown in **Fig. 2.13** is the inverted image stack after segmentation of **Fig. 8.7**.

The fracture can be further accentuated using 'Merge Channels' (Image → Color → Merge Channels) which allows the fracture outline to be highlighted in red while the rock background remains in grayscale (**Fig. 2.14**). This superimposed image stack can be viewed in 3D Viewer as its own orthoslice or volume or added to an already existing image such as the fracture volume. The latter operation can be accessed from the 3D Viewer window (Add → From Image).

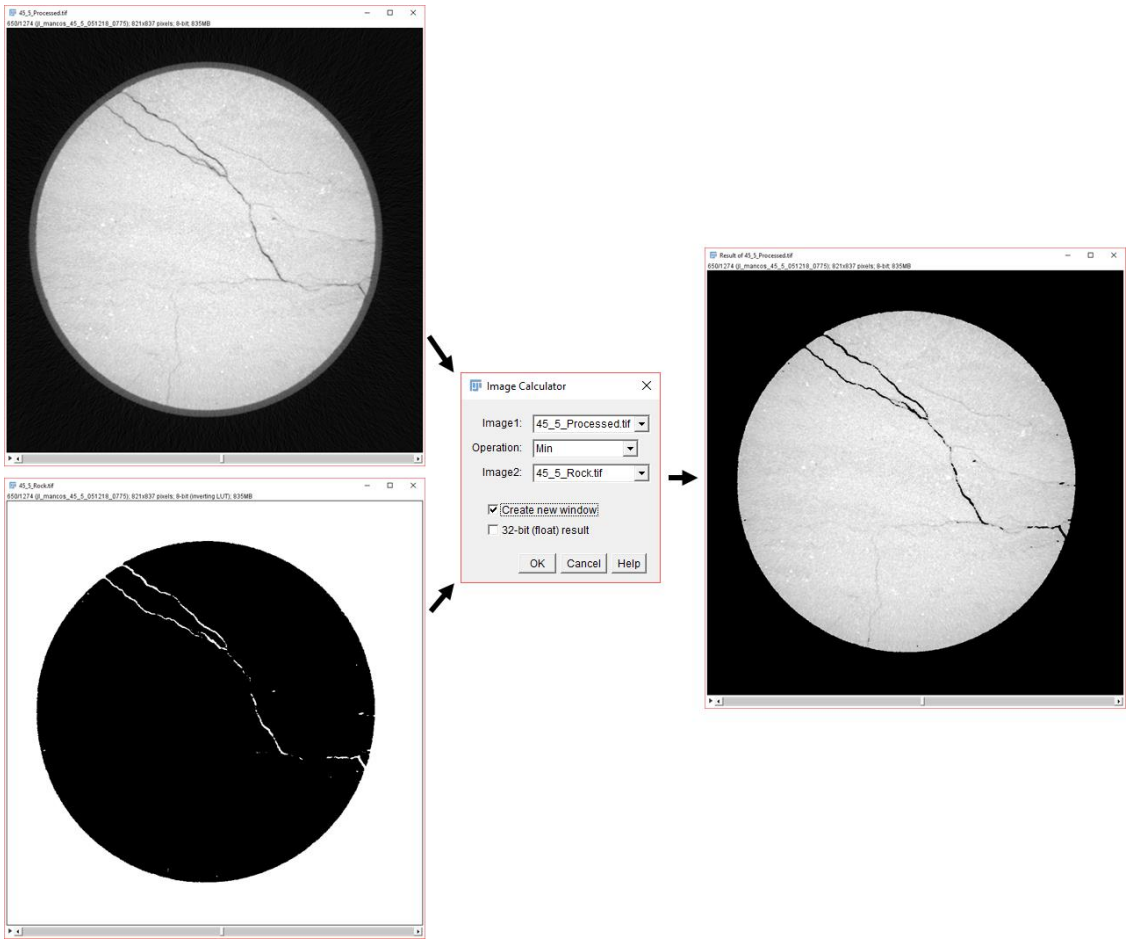

Fig. 2.12—Superimposing segmented fracture onto background rock using Image Calculator.

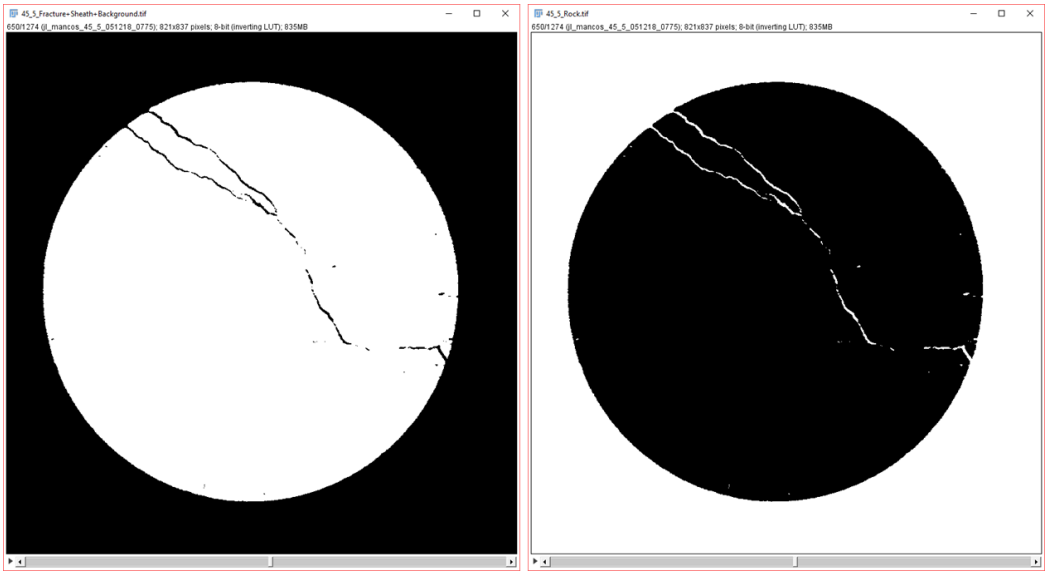

Fig. 2.13—Isolated fracture stack (left) and stack after inversion (right).

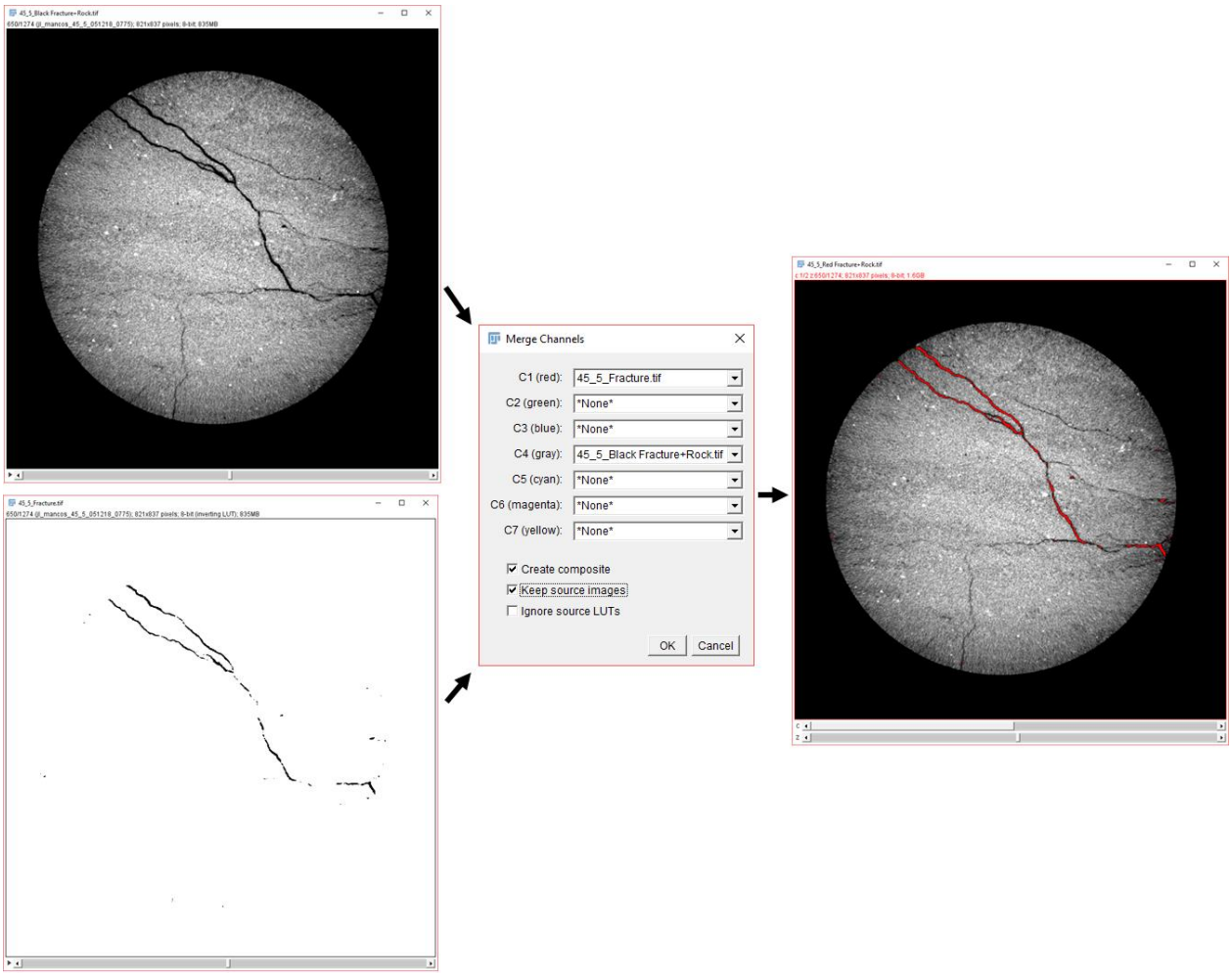

Fig. 2.14—Merging segmented fracture with superimposed stack to further accentuate fracture network.

2.3 Results

2.3.1 Perpendicular Bedding at 0°

Fig. 2.15 shows the stress loading results. From the plots, loading Young's modulus of 0.90 MMpsi, unloading Young's modulus of 2.08 MMpsi, and Poisson's ratio of 0.086 were calculated.

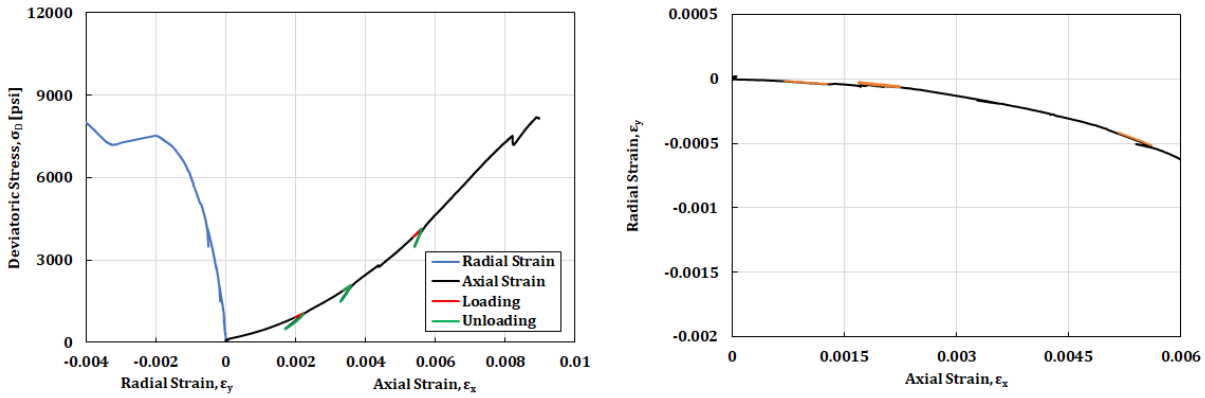

Fig. 2.15—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right).

Fig. 2.16 shows the interpreted fracture network superimposed on the background rock orthoslice and volume from different orientations for sample PD_15. The dominating fracture plane is vertically orientated with some branching fracture orientated at 45°. Layering did not appear to have a noticeable effect on fracture propagation and orientation.

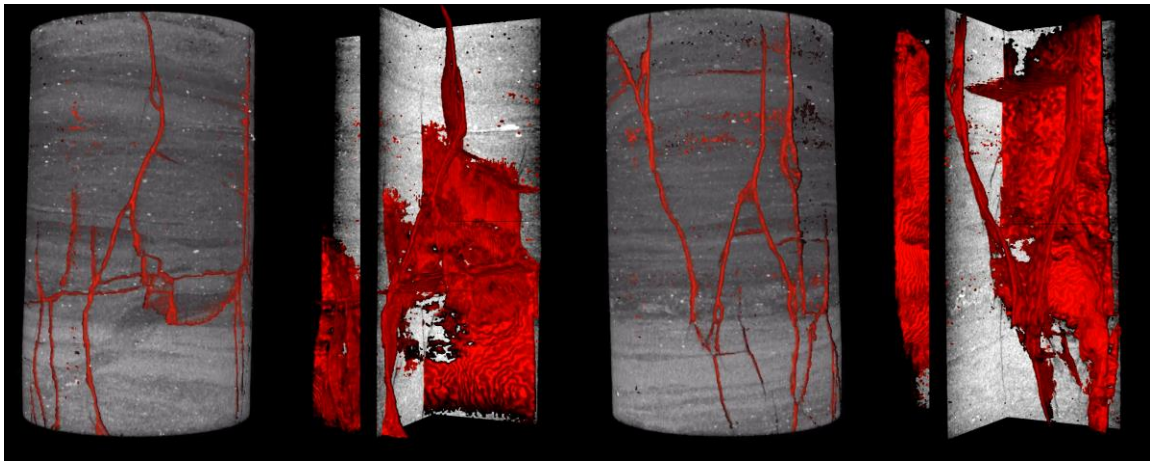

Fig. 2.16—Fracture network shown in red superimposed on rock volume (left) and orthoslice (right).

2.3.2 Direction Bedding at 45°

Fig. 2.17 shows the stress loading results. From the plots, loading Young's modulus of 1.21 MMpsi, unloading Young's modulus of 2.29 MMpsi, and Poisson's ratio of 0.083 were calculated.

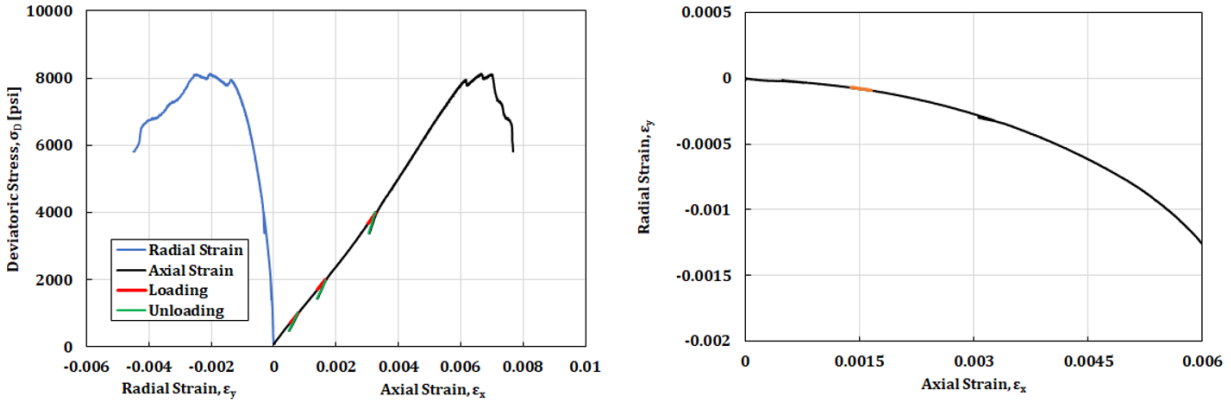

Fig. 2.17—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right).

Fig. 2.18 shows the interpreted fracture network superimposed on the background rock orthoslice and volume from different orientations for sample 45_5. The dominating fracture plane is orientated at 45° and coincides with the bedding orientation. Fractures mostly propagates along the interface between the darker facie and the lighter facie.

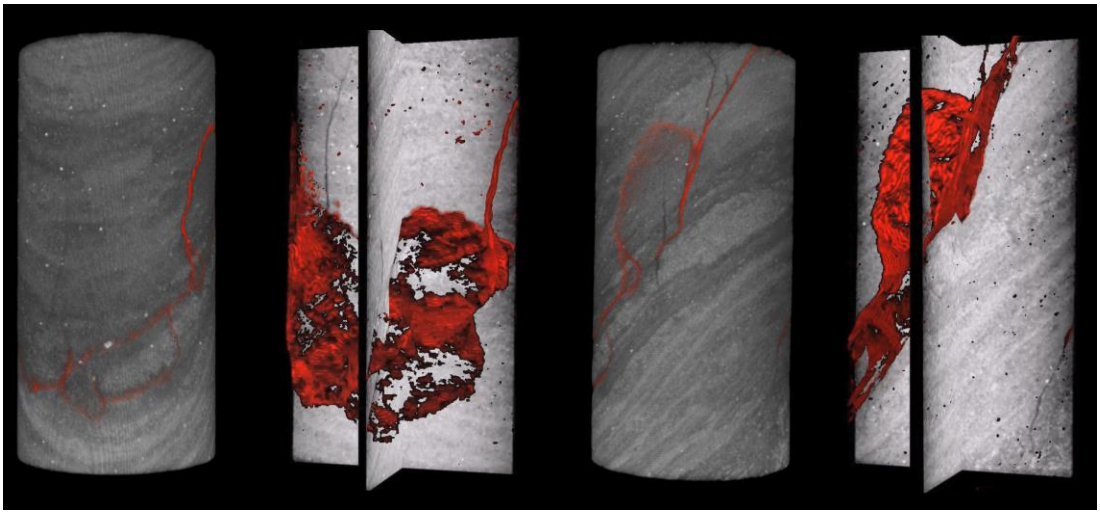

Fig. 2.18—Fracture network shown in red superimposed on rock volume (left) and orthoslice (right).

2.3.3 Parallel Bedding at 90°

Fig. 2.19 shows the stress loading results. From the plots, loading Young's modulus of 1.58 MMpsi, unloading Young's modulus of 2.38 MMpsi, and Poisson's ratio of 0.166.

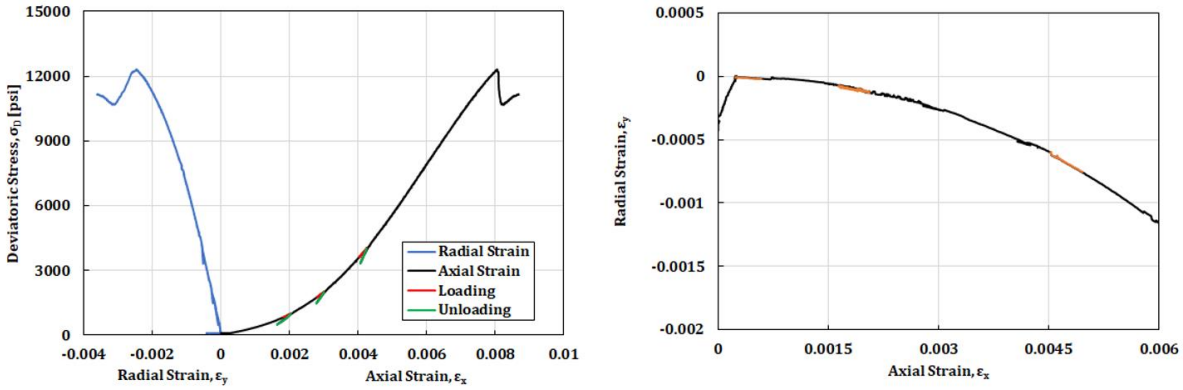

Fig. 2.19—Deviatoric stress plotted against strain (left), and radial strain against axial strain (right).

Fig. 2.20 shows the interpreted fracture network superimposed on the background rock volume from different orientations for sample PL_6. The dominating fracture plane is oriented near 90° and coincides with the bedding orientation. The most prevalent branching and independent fractures are also oriented near vertical.

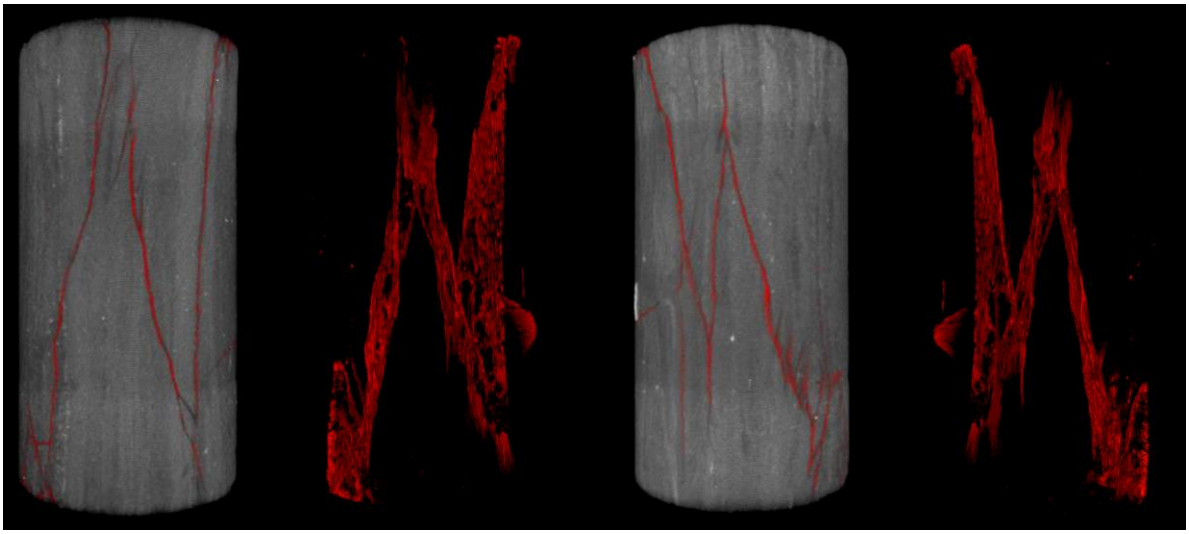

Fig. 2.20—Fracture network shown in red (right) superimposed on rock volume (left).

2.4 Discussion

2.4.1 Effect of Bedding Orientation and Pre-Existing Fractures

While the complexity of the fracture networks varied widely over the samples, the majority of the samples shared several common characteristics. A dominant fracture plane was usually visible though in some cases its identification was complicated by the presence of numerous branching and independent fractures. The distinction between branching and independent fractures is simple: for those fractures that do not branch off from the dominant fracture plane are denoted as "independent fractures". Also, for the majority of the samples, the number of branching fractures outnumbered the number of independent fractures.

Table 2.1 summarizes the fracture network characterization of 9 samples. For most of the samples, fractures developed along the same orientation of the lamination planes. This is especially true for samples with directional bedding at 45° in which the major fracture plane and the majority of the branching fractures aligned at 45°. For perpendicular and parallel bedding samples, fracture orientation was more unpredictable. Parallel bedding samples recorded the greatest number of independent fractures. The orientation of "independent fractures" did not strongly correlate with the orientation of either the bedding orientation or pre-existing fracture orientation.

Table 2.1—Fracture network characterization summary table.

Sample Name	Samples Intact?	Bedding Orientation	Fracture Orientation[1]	Branching Fractures[2]	Independent Fractures[3]
45_1	Yes	45°	45°	3	1
45_2	Yes	45°	45°	1	0
45_5	No	45°	45°	5	2
PD_11	Yes	0°	45°	1	0
PD_15	No	0°	90°	7	1
PD_17	Yes	0°	45°	1	0
PL_6	Yes	90°	90°	3	1
PL_7	No	90°	45°	4	4
PL_16	Yes	90°	45°	3	3

[1]The orientation of the dominant fractures.
[2]The number of dependent fractures branching off of the dominant fracture.
[3]The number of independent fractures not connected to the dominant fracture.

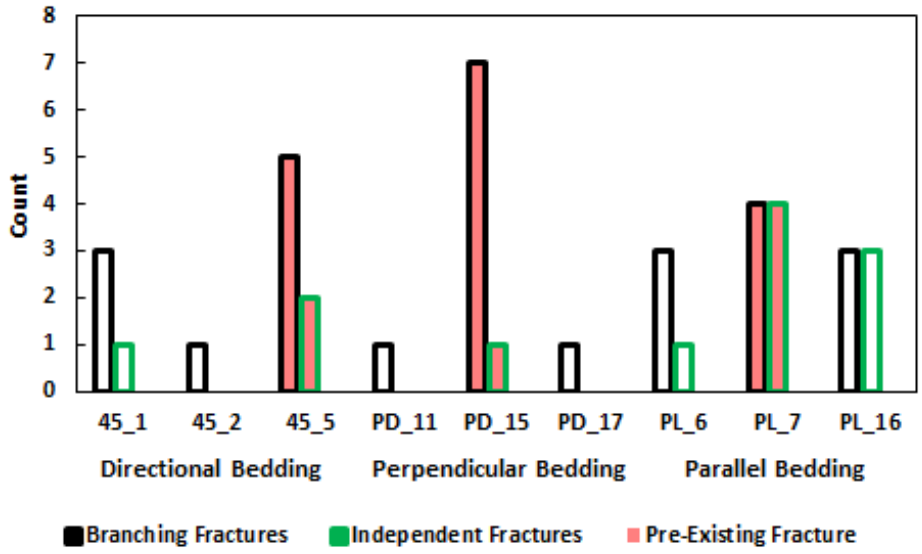

Fig. 2.21—Fracture network shown in red (right) superimposed on rock volume (left).

With intact samples, induced fractures preferentially aligned along bedding planes (**Fig. 2.22**). With pre-existing fractures, Ramos (2018) verified an increase in sample compliance. Induced fractures preferentially aligned along failure planes of pre-existing fractures (**Fig. 2.23**). However, because the orientation of most pre-existing fractures corresponded to the orientation of bedding planes, we couldn't determine which factor had a greater influence in determining the orientation of induced fractures. The rows highlighted in red of Table 7 were the samples with pre-existing fractures. As expected, these samples recorded a higher number of branching fractures (**Fig. 2.21**). This verifies similar results by Suarez-Rivera et al. (2013) in which they determined that the location and alignment of new fracture planes was primarily influenced by bedding and pre-existing fractures.

While the number of branching fractures certainly increases with pre-existing fractures, the way these branching fractures orient is hard to predict. The irregular nature of the branching fracture orientations and the presence of independent fractures not connected to the dominant fractures highlight the complex stress state that develops along pre-existing fractures. We surmise that it is ultimately this complex stress state that dictates whether fractures occur along a pre-existing fracture path or branch out to complete the failure plane (Ramos 2018).

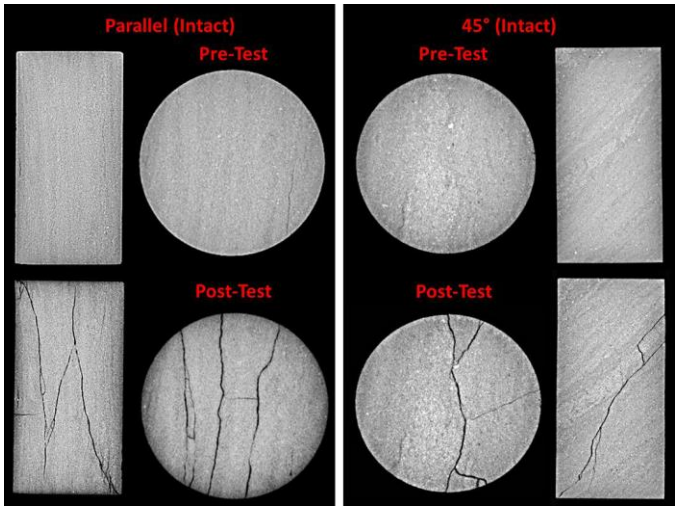

Fig. 2.22—Parallel bedding sample (PL_6, left) and 45° bedding sample (45_1, right) (Ramos 2018).

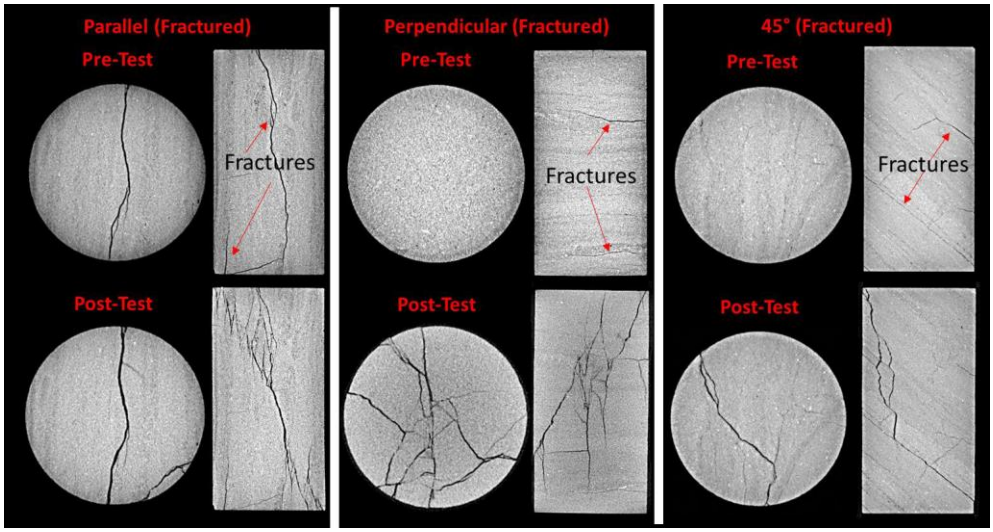

Fig. 2.23—Parallel bedding sample (PL_7, left), perpendicular bedding sample (PD_15, middle), and 45° bedding sample (45_5, right) (Ramos 2018).

2.4.2 Effect of Bedding Orientation and Confining Stress

Ramos (2018) found that increasing deviatoric and confining stresses decreased the anisotropic radial deformation of parallel bedding samples. At low confining stress, radial expansion occurred in the direction perpendicular to bedding. As a result, most fractures develop along the same orientation of the lamination planes. More of these fractures were of the branching variant than of the independent type, thought parallel bedding samples showed a higher count of independent fractures (**Fig. 2.24**).

In terms of the effect of confining stress, the greater the confining stress, the lower the count of both branching and independent fractures. This was expected since increasing confining stress leads to increasing sample compliance. Those samples that were hydrostatically loaded at 12 ksi and brought to failure at 3 ksi recorded the lowest count of branching and independent fractures.

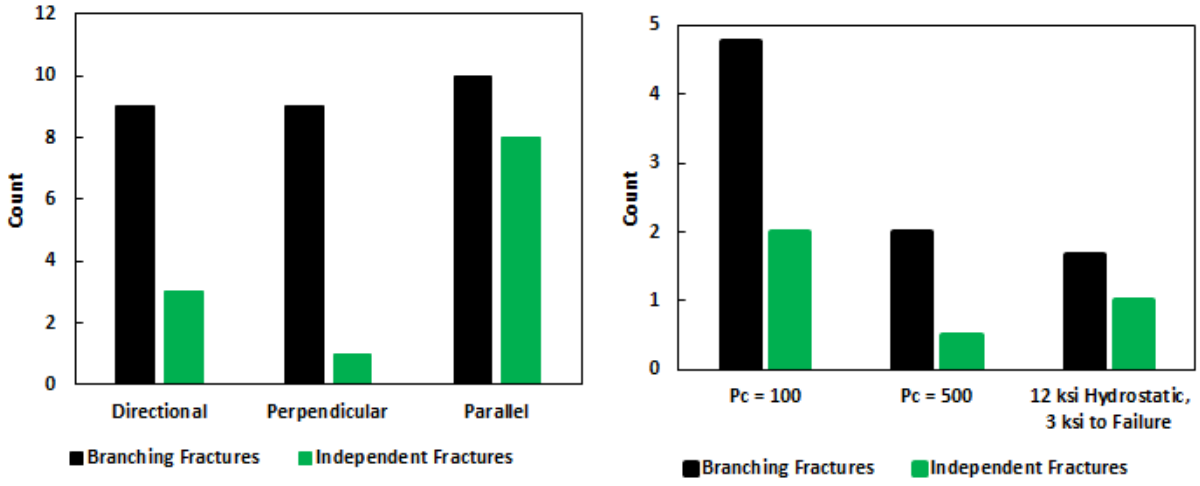

Fig. 2.24—Fracture network shown in red (right) superimposed on rock volume (left).

2.4.3 Limitations in Identifying Fractures through Image Segmentation

Like previously mentioned, in order to visualize the fracture network the fracture must be first segmented through thresholding. However, because the inspection of very fine fracture networks within larger specimen samples requires scanning at coarser resolutions in order to resolve the sample in its entirety, very fine fractures cannot be accurately segmented and visualized at these resolutions. Moreover, existing limitations of the X-ray technique such as beam-hardening, ring artifacts, and blurring further complicate fracture segmentation. This section overviews some of the techniques used to address these issues.

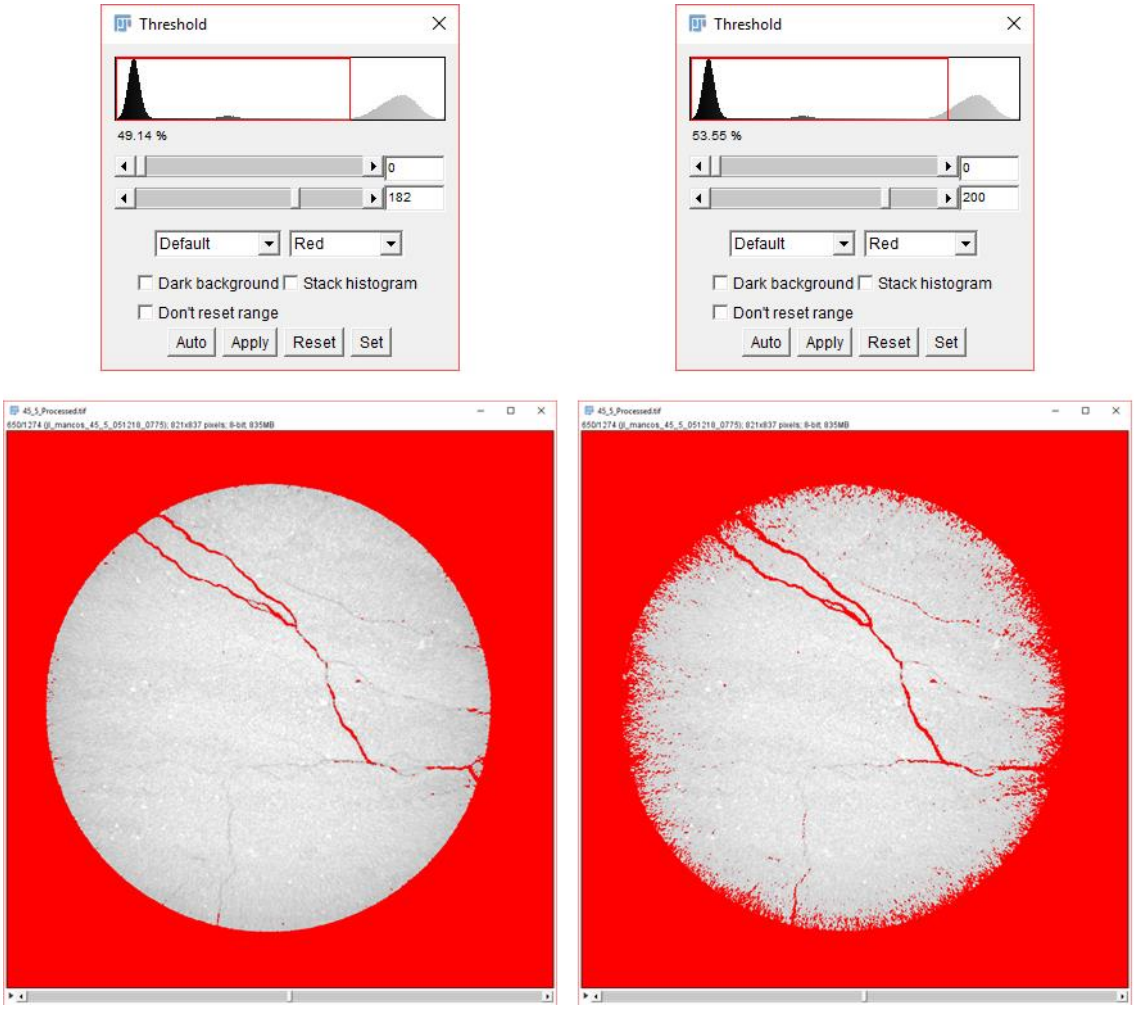

Fig. 2.25—Segmentation results from two different threshold limits. The greater the limit the more noise.

When using the global thresholding technique, too high of a threshold limit results in slightly better fracture segmentation but at the cost of more noise localized around the edges of the sample (**Fig. 2.25**). This is due to imaging artefacts caused by blurring and possible overcorrection of beam-hardening effects. CT blurring is especially evident with smaller aperture traverses. For very thin apertures, the CT number anomaly used to distinguish fracture from the matrix background is barely distinguishable from the matrix noise (Ketcham et al. 2010, Vandersteen et al. 2003, Van Geet et al. 2001). This makes the segmentation process less accurate, especially when resolving these features at coarser resolutions.

Beam hardening artifacts occur due to the differential attenuation of high energy and low energy X-rays (Ketcham et al. 2001). Low energy X-rays are more likely to be stopped than high energy X-rays as these X-rays travel through the sample. This differential attenuation effect manifests itself as bright rims and dark centers of objects. The outer surface of the sample appears brighter (more dense) as more lower energy X-rays are being stopped in this area relative to the center. Beam hardening can be automatically corrected using preset algorithms available on the reconstruction software CTPro3D. Based on the severity of beam hardening, the user can select the most suitable preset to reduce the beam hardening ring without losing too much detail. **Fig. 2.26** shows the results after 3 different presets are applied. Each preset has different coefficients for the polynomial formula used to correct for beam hardening. The subjective nature of the correction is the largest drawback to this type of beam hardening reduction technique. The user must subjectively determine which selection preset best reduces the beam hardening ring. The third preset was chosen for all the reconstructed scans of this study.

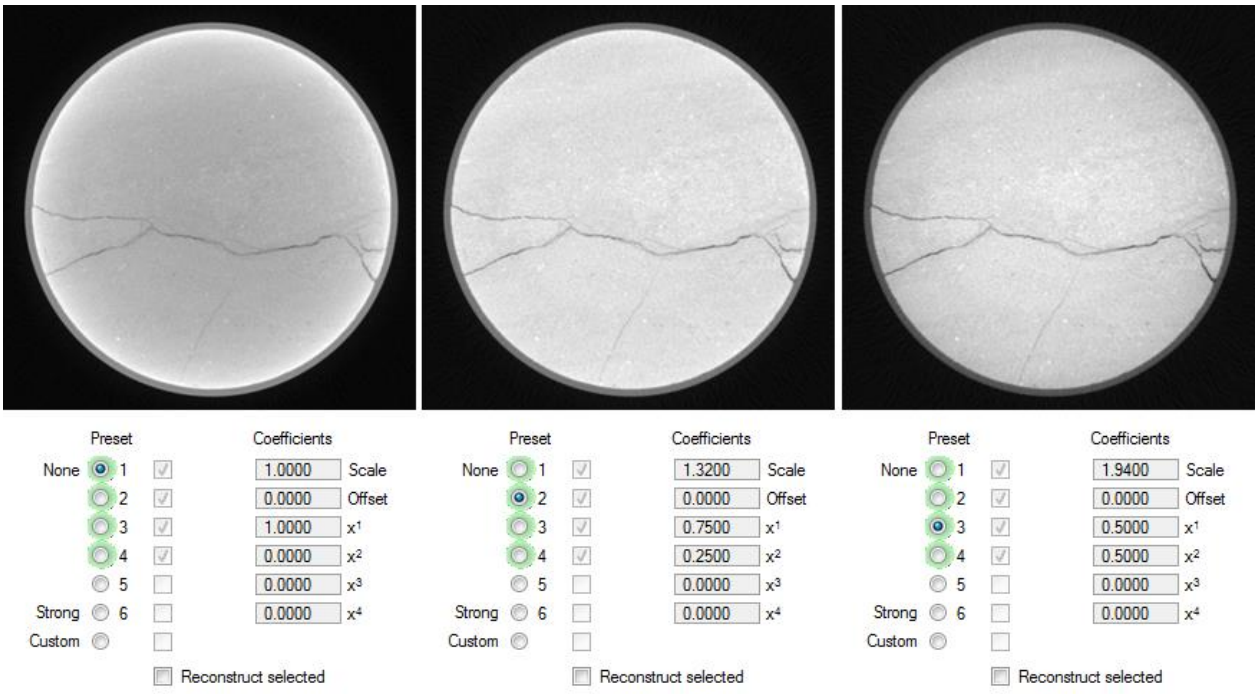

Fig. 2.26—Beam hardening correction results from three different presets.

Over-correction of the beam hardening effect can actually have opposite of the desired effect. For example, too much over-correction of the brighter outer surface of the sample can actually make the outer surface of the sample too dark. This could explain why in **Fig. 2.25** as one increases the threshold limit, the outer surface of the sample with its elevated grayscale number is incorrectly segmented with the fracture volume. The effect of overestimating the threshold limit on the 3D visualization of the fracture network is shown in **Fig. 2.27**. The fracture network interpretation with a lower threshold limit is shown in the left column in red. Applying a higher threshold limit leads to slightly better fracture segmentation but at the cost of more noise localized around the edges of the sample. The 3D results are shown in the middle column in green. There is so much noise that it masks the actual fracture network. The noise concentrated around the outer circumference of the sample can be cropped out to give the results shown in the right column.

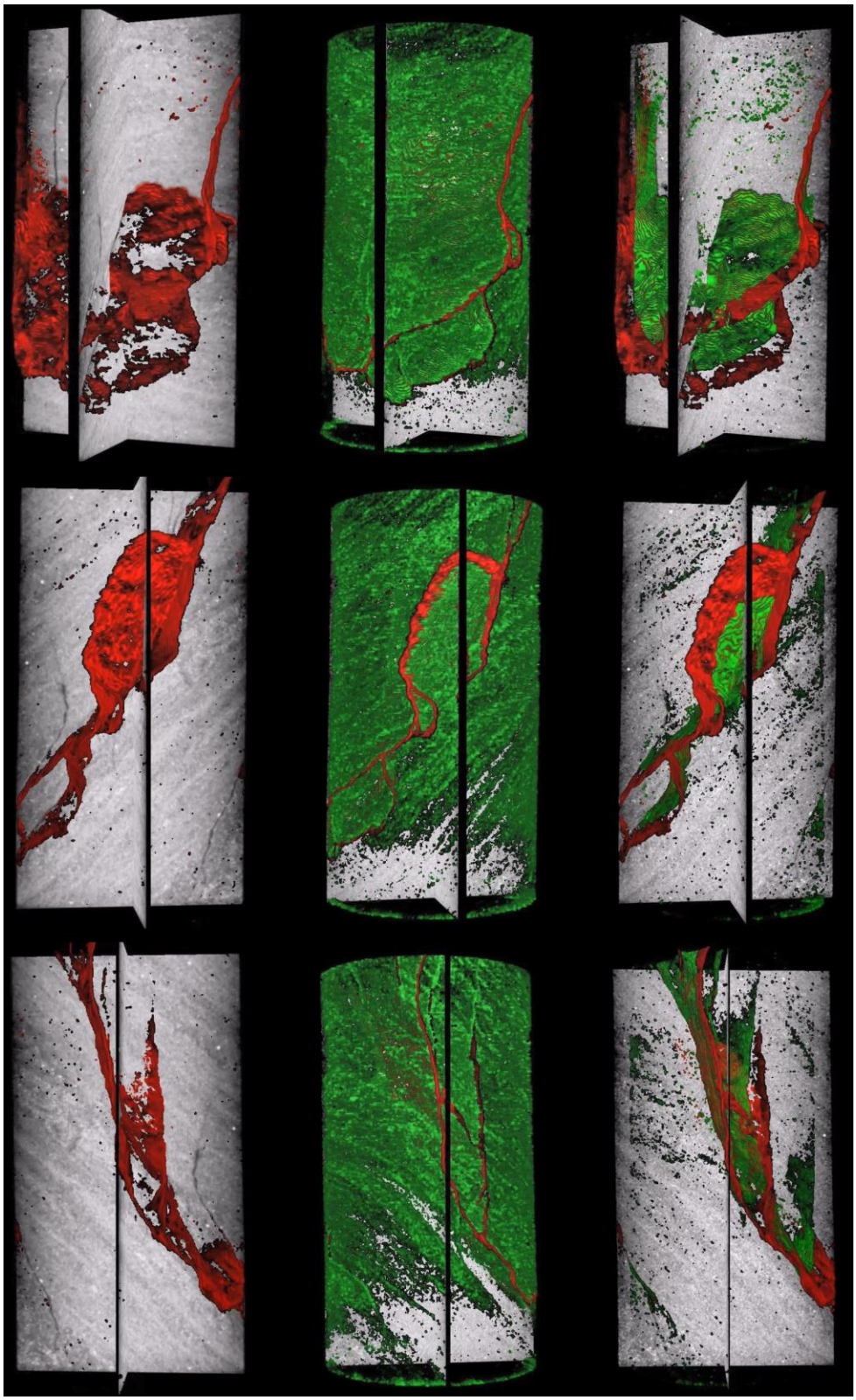

Fig. 2.27—Original fracture network (left), with higher threshold limit (middle), and after cropping (right).

There exists more sophisticated segmentation algorithms besides the simple, global thresholding variant type, including statistical types (k-means clustering), region growing/level set based types, and those that adapt to local image information (indicator kriging). Manual segmentation is possible but completely infeasible for 3D volumetric images. For this study, statistical region merging was tested but to no avail. Indicator kriging segmentation shows the most appeal as it combines the global and local information of the grayscale values for segmentation. While not applied for this study, Wang (2018) has made headway in successfully segmenting and quantifying the fracture network of a Bakken shale sample using entropy-assisted, indicator kriging method.

2.5 Conclusions

In this study, we utilize X-ray micro-computed tomography (CT) imaging to map fracture networks of shale samples in an attempt to understand the potential for pre-existing fractures and rock heterogeneities to aid or hinder the propagation of fractures induced. Based on our results, the following conclusions can be drawn:

- Both intact samples and samples with pre-existing fractures show that induced fractures tend to develop along the same orientation of lamination planes, which more often than not correspond to the same orientation of any pre-existing fractures.
- While simple, global thresholding can effectively segment the dominant fracture networks, finer features are not resolved due to insufficient image resolution and existing physical limitations of the X-ray technique, including beam-hardening, ring artifacts, and blurring.

Chapter 3: X-Ray Micro-CT Observation of Methane Hydrate Growth and Dissociation in Sandy Sediments

Jeffery S. Luo, Xiongyu J. Chen, Tiannong Dong, D. Nicolas Espinoza, and David A. DiCarlo

3.1 Introduction

While much focus has been devoted to unconventional production of hydrocarbon resources in the US due to the recent shale revolution, research into other hydrocarbon sources such as methane hydrates is still important due to the inevitability of resource withdrawal and decline. Methane hydrates are especially appealing since its total estimated carbon amount is on the same order of magnitude of all fossil fuels combined (Klauda and Sandler 2005; Kvenvolden 1988). While distributed in vast amounts in continental margins and permafrost regions, much remains unknown in terms of how the presence of methane hydrates in pores changes the flow characteristics and physical properties of these complex systems. Understanding these properties is of fundamental importance if the goal is to be able to one day commercially produce from these vast methane hydrates deposits.

The pore-habit of hydrate within sediments is especially important since it directly affects the relative permeability of hydrate bearing sediments (Keehm and Yoon 2008; Kang et al. 2016). Dai and Seol (2014) used pore network modeling methods to determine relative permeability in hydrate bearing sediments. Their work showed that the relationship between relative permeability and hydrate saturation depends mainly on type of pore habit (grain-coating, pore-filling, or dispersed pore-filling) and grain size distributions. Kneafsey et al. (2011) conducted experimental corefloods and measured hydrate saturation using a medical X-ray CT scanner. They found that a pore-filling model is most representative of their measured gas relative permeability data. Kang et al. (2016) showed that the hydrate pore habit changes from grain-coating to pore-filling around a hydrate saturation of 20 to 40%.

Recently, Chen (2017) used an X-ray CT scanner to image xenon hydrate growth in sand and extracted 3D pore structures of dry and hydrate-bearing sand. Based on these 3D pore structures of dry sand, he also generated computational models with idealized hydrate pore habits. In disagreement with previous studies (Kang et al., 2016; Kumar et al., 2010), Chen found it hard to distinguish between the transition from grain-attaching to pore-filling habits from just $k_{rg} - S_{hyd}$ data. In addition, Chen found that dispersed pore-filling habit, in which porous hydrate crystals reside in the pore space, significantly decreased k_{rg} by up to one order of magnitude when compared with either of the patchy hydrate habit models (grain-attaching or coarse pore-filling). An illustration of the three major types of pore-habit models is shown in **Fig 3.1** (Chen et al. 2017)

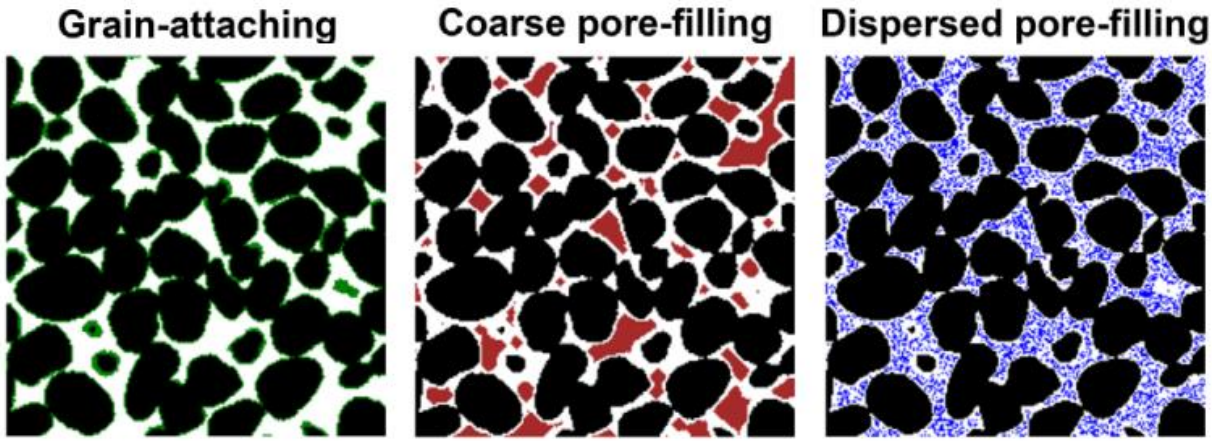

Fig. 3.1—Illustration of the three major types of pore habit models showing hydrate saturation of ~30%

Chen continued studying the evolution of hydrate pore habit and spatial distribution of hydrate bearing sediments with more experimental work (Chen and Espinoza 2018b). He showed experimental evidence of Ostwald ripening of gas hydrate crystals in pores and found that Ostwald ripening gradually changes hydrate pore habit from grain-attaching to pore-filling. As the pore habit changes to pore-filling, Chen observed increasing heterogeneity with more patchy hydrate distribution and spatial variation in permeability and sediment strength.

Chen's most recent work looked at the effect of water salinity and aqueous concentrations of methane in water on methane hydrate stability (Chen et al. 2018a). His excess-gas experiments showed that water mobilized over fairly large distances even at water saturation below 30% to facilitate hydrate ripening and three-phase equilibration. This water movement explained the resulting heterogeneous distribution of hydrate and interconnected pore habit. Chen also showed that local brine salinity changes as a result of methane hydrate growth and such salinity change can in turn affect the hydrate stability zone.

The work shown in this chapter is an extension of Chen's experiments. Much credit is owed to Chen who originally designed the micro-consolidation device used in this project. Chen's experiments were of the excess-gas variant while the experiments conducted in this work is of the excess-water variant. In the excess-gas method, a known volume of water is mixed with the sand prior to hydrate formation. Hydrate saturation is restricted by the amount of added water. Ideally, pressure will gradually decrease as saturation of hydrate increases, but will level off as finite water is consumed. In the excess-water method, a known volume of gas is injected into the specimen prior to hydrate formation. Water is subsequently injected until a target pore pressure within the specimen is achieved. Hydrate saturation is restricted by the amount of added gas. Ideally, water intake will gradually increase as saturation of hydrate increases, but level off as finite methane is consumed (Priest 2009).

In this work we look at the influence of hydrate formation conditions, both excess-gas and excess-water, on hydrate pore habit and the spatial distribution of hydrate bearing sediment. We show pore-scale coexistence of brine, gas, and hydrate with unambiguous phase segmentation and the evolution of these phases as brine salinity increases and hydrate growth continues. We also monitor hydrate dissociation in order to better interpret any production tests from these hydrate bearing sands.

3.2 Materials and Methods

3.2.1 Rock and Fluid Types

Two methane hydrates experiments are conducted in coarse Ottawa sand with a median grain size of 700 μm under excess-water conditions. As opposed to excess-gas conditions, there exists enough water to take all available brine to three-phase brine-gas-hydrate stability. 4.4 wt% brine are used in both experiments.

3.2.2 PID Thermoelectric Cooling and Data Logging

3.2.2a Experimental Setup

Methane hydrates are synthesized in a modified vessel originally created by Dr. Xiongyu Chen (2018) at controlled temperatures and pressures with a suite of electrical and hardware components (**Figs. 3.2** and **3.3**). Thermistors attached to the insulated sample vessel measure temperature. Temperature readings are read as digital inputs into an Arduino microcontroller. A PID algorithm programmed within the Arduino integrated development environment (IDE) regulates two Peltier cooling assemblies to maintain target temperature, which can be set as low as 1 °C. Thermistor (T1) measures temperatures at both the top of the Peltier assembly and the top of the sample vessel while thermistor (T2) only measures temperatures at the bottom Peltier assembly. Thermistor (T3) records temperatures at the bottom of the sample vessel. Only T1 and T2 are used in the PID algorithm to control thermoelectric cooling.

The cold side of the Peltier cells are oriented towards the ends of the sample vessel where thermoelectric cooling is to take place. The hot side of the Peltier cells are oriented away from the vessel. Two heat sinks transfer the heat generated from the hot sides of the Peltier cells to separate fans where heat is dissipated. Thermal pads with thermal conductivity of 6 W/mK coat both sides of the Peltier cells.

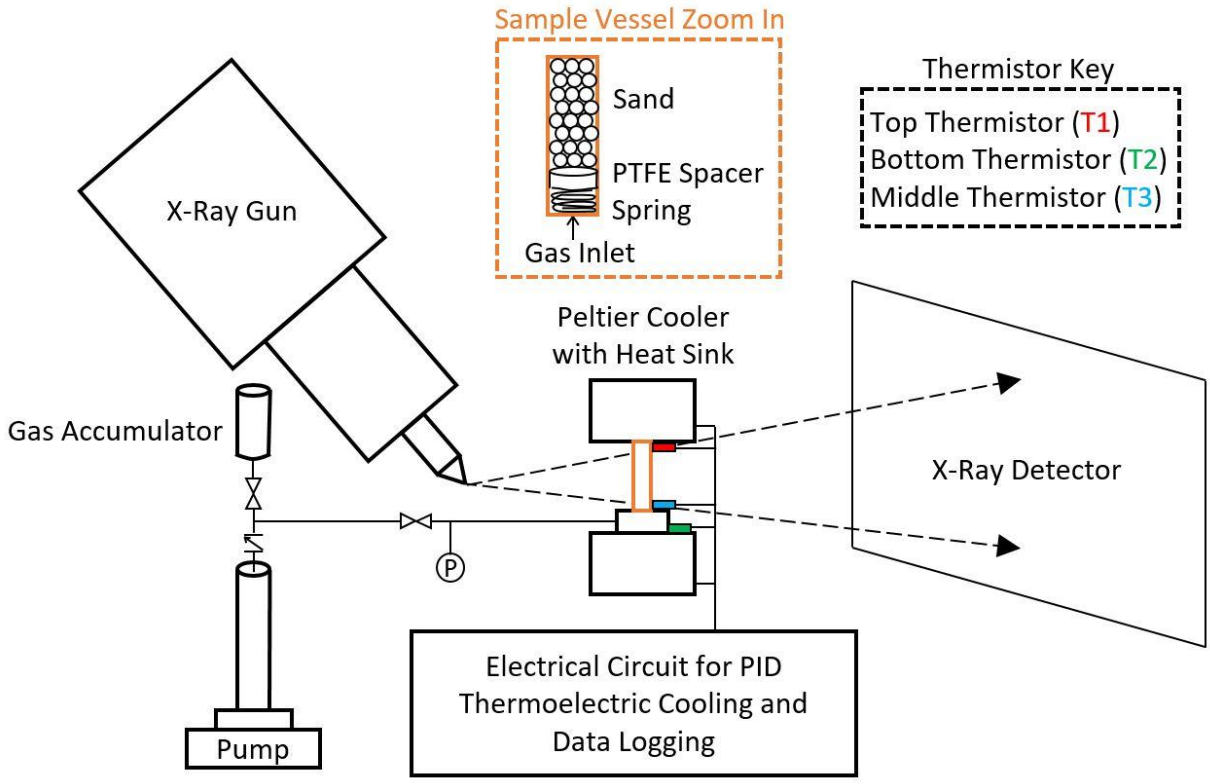

Fig. 3.2—Schematic of experimental setup with micro-consolidation cell and thermoelectric cooling capability.

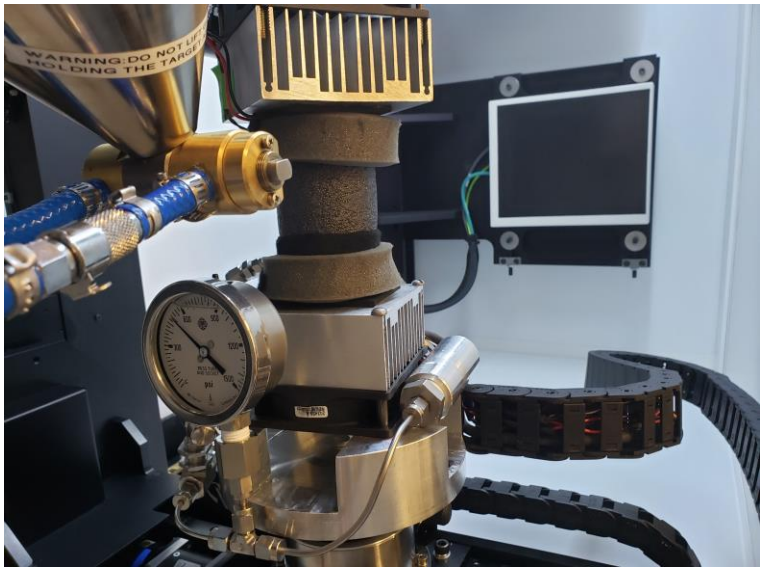

Fig. 3.3—Picture of experimental setup, electrical ensemble out of frame and shown separately in Fig. 3.5.

Within the aluminum sample vessel of inner diameter 7.9 mm and length 31.3 mm, Ottawa sand with a grain size of 0.7 mm is packed with PTFE/aluminum spacers and stainless steel sieves. A stainless steel precision compression spring of outer diameter 0.24″, wire diameter of 0.022″, and free length of 0.5″ subjects the sandpack at a loading rate of 8 lbf/in. Approximately 90 sand particles are packed into a circular sandpack layer (Specht, 2018). With the amount of sand added known, the height of the sandpack is calculated from an estimation of the total number of sand particles in the vessel. The compressed length of the spring is the difference between the spring's free length and the available space within the vessel after accounting for the sandpack, spacers, and sieve height. The applied axial stress is the product of the compressed length of the spring with its loading rate. This configuration allows for a maximum axial stress of 75 psi.

Fig. 3.4 shows the electrical ensemble. Data logging through the addition of a micro-SD breakout board and DS 3231 real time clock module allows for constant monitoring of temperatures and pressures (Karlsen, 2010). Temperatures and pressures are also conveniently displayed on a LCD screen. Breadboard diagrams of the electrical components are shown in **Figs. 3.5 and 3.6**. The former is of an electrical ensemble suited for the Arduino Mega, the latter for the Arduino Uno. A LCD display is not included in the Arduino Uno variant due to lack of available digital input pins.

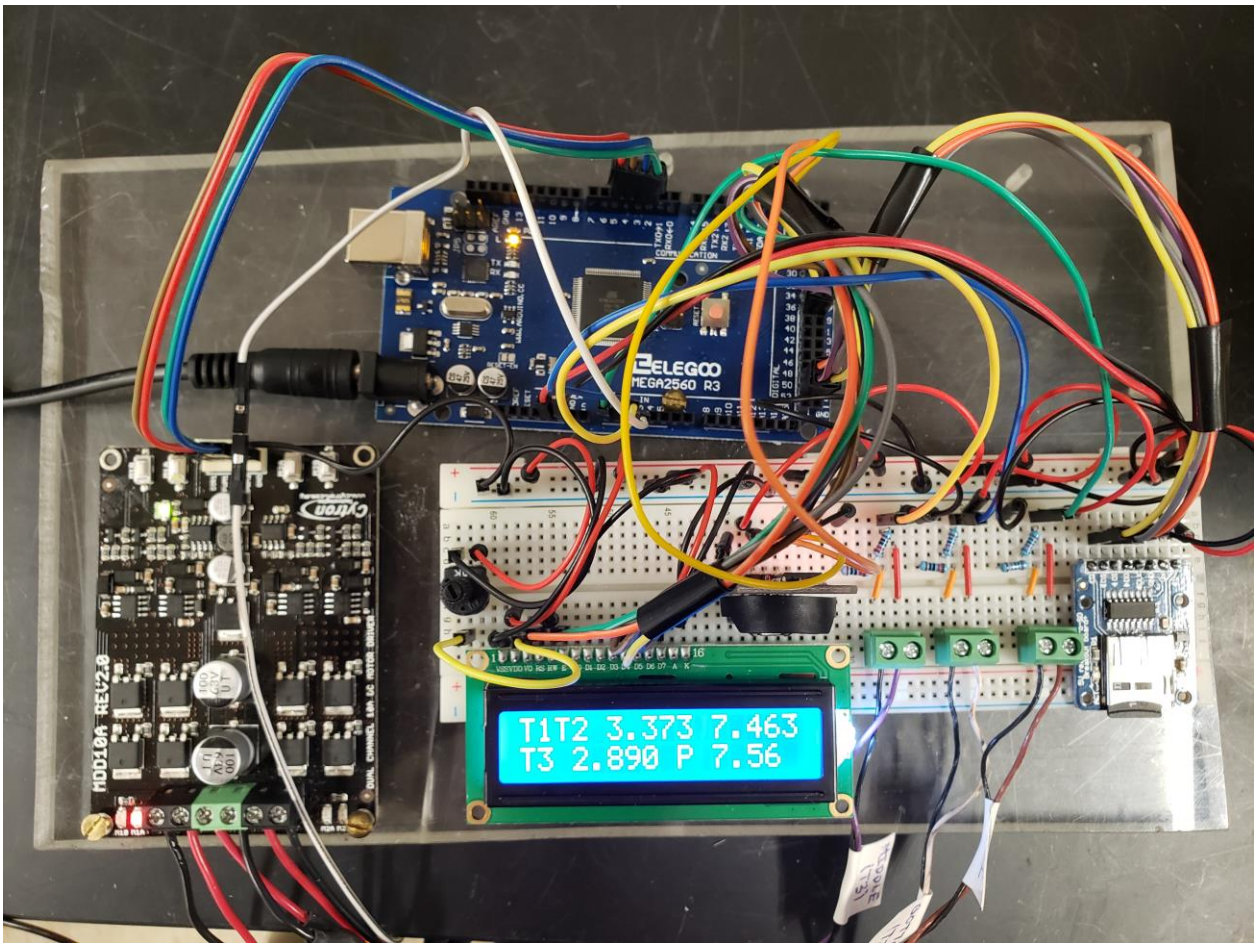

Fig. 3.4—Picture of electrical components for experimental setup

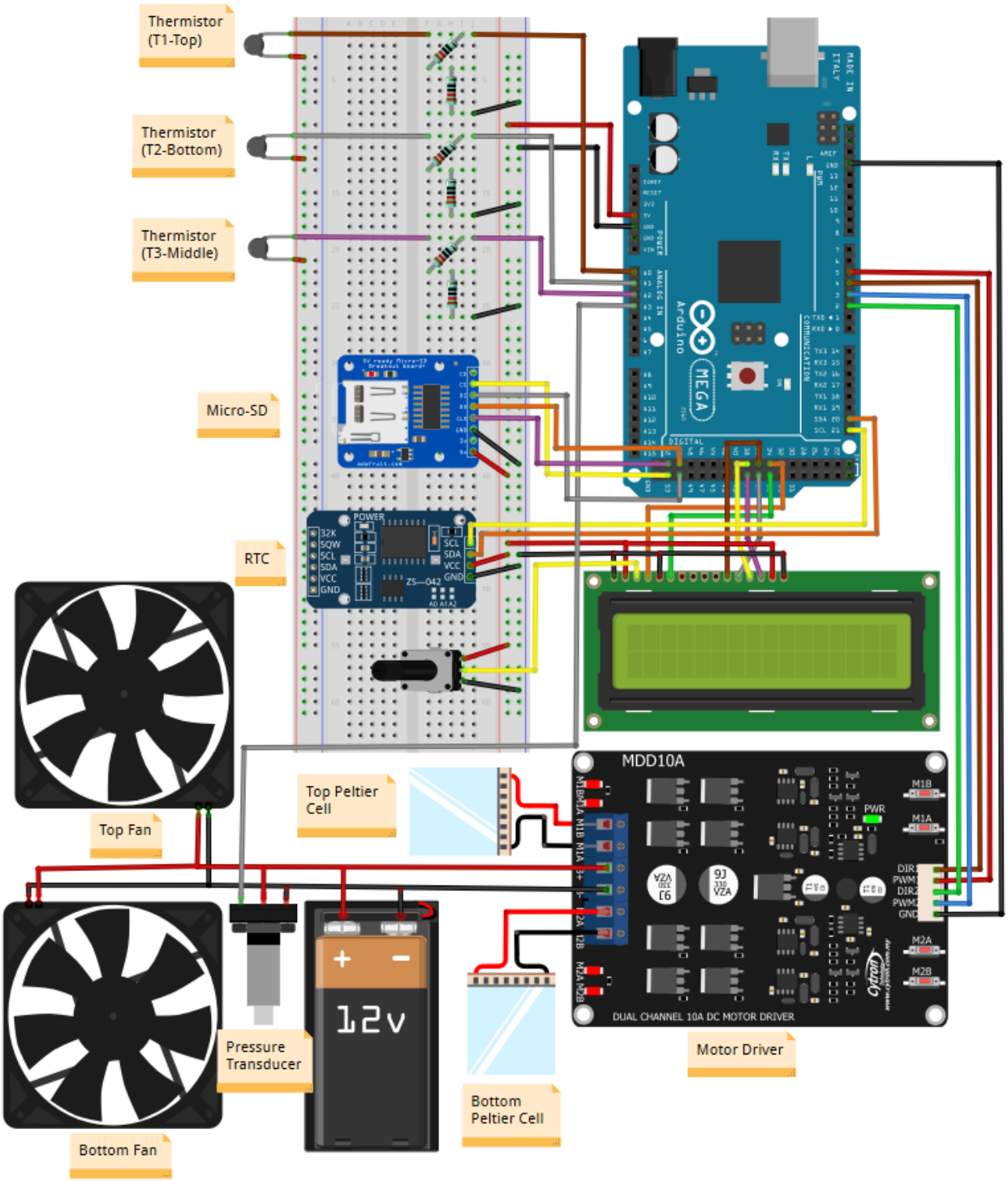

Fig. 3.5—Fritzing diagram of electrical ensemble for methane hydrate generation (Arduino Mega variant)

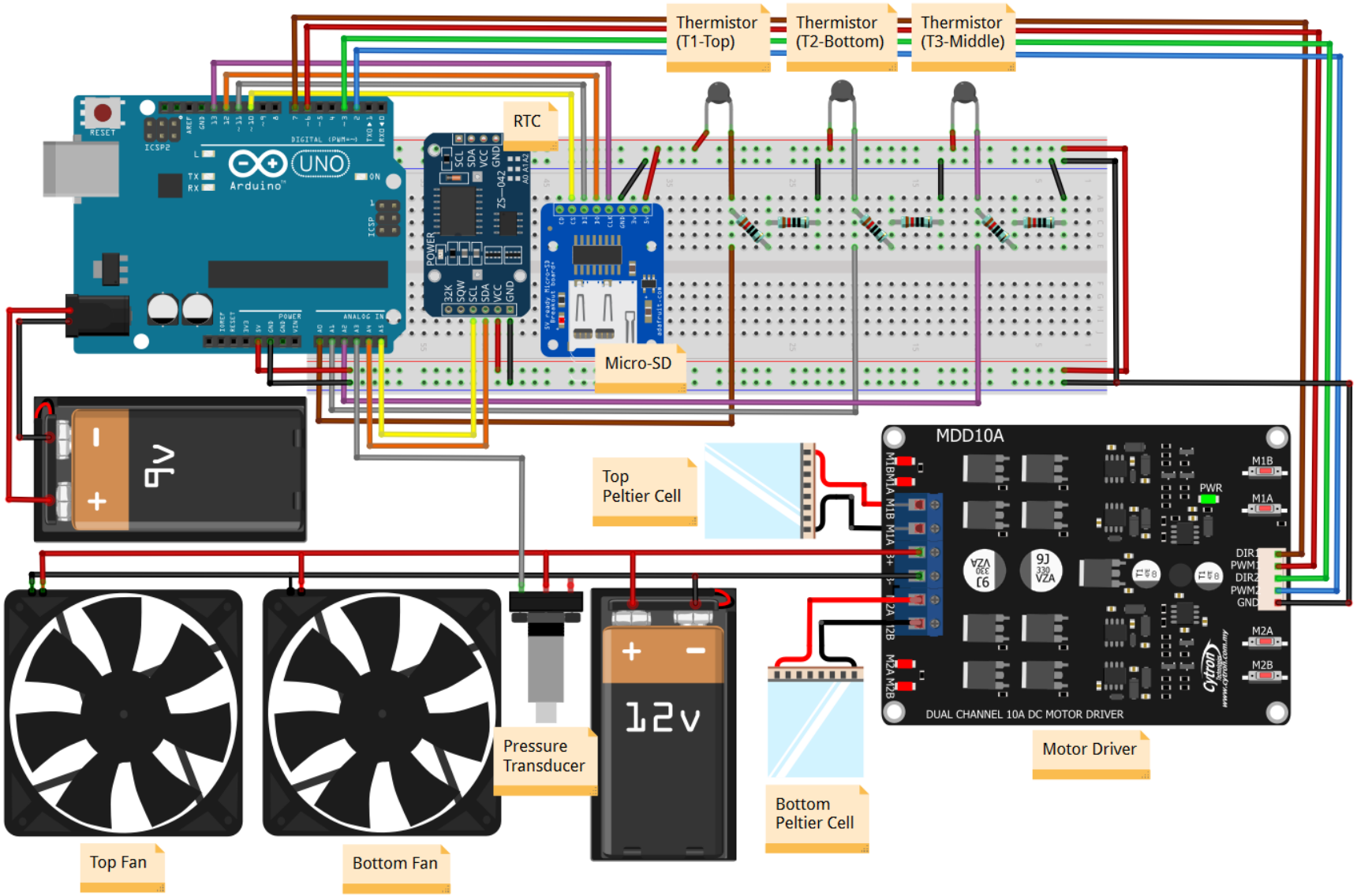

Fig. 3.6—Fritzing diagram of electrical ensemble for methane hydrate generation (Arduino Uno variant)

3.2.2b Hydrate Formation

For excess-water experiments, the only open end of the vessel is connected to a methane gas accumulator and ISCO pump. First, methane gas is slowly injected into the pore space until a predetermined methane pressure is reached. This initial gas pressure is based on the amount of gas hydrate desired (Priest, 2009). Details of this calculation are described in the following section 3.2.3c. Once the desired gas pressure is reached, methane gas supply is locked off. Vessel temperature is brought down to approximately 3 °C. De-aired water is then injected into the vessel using an ISCO pump until a predetermined water pressure is reached (~1300 psi), thereby causing the pressure-temperature conditions within the vessel to cross the hydrate-stability boundary and initiate methane hydrate formation.

The ISCO pump is ran on constant pressure mode since pressure will decrease as methane is consumed and converted to hydrate. The only instances in which the device is not kept at constant pressure is when the micro-consolidation device is scanned using micro-CT since the ISCO pump cannot be placed or powered from within the X-ray cabinet. Ideally a syringe pump similar to the one detailed in Appendix C should be used to maintain constant pore pressure during scanning though such a pump would have to be rated for especially high pressures. The pump shown in Appendix C is a low pressure pump rated to approximately 100 psi. After micro-CT scanning (usually 25 minutes in duration), the device is re-connected to the ISCO pump and brought back onto constant pressure mode operation. The experiment is allowed to run for several days to ensure full conversion of free gas to methane hydrate. At this point no more water should be required to maintain constant pore pressure.

3.2.2c Initial Gas Loading Pressure

For excess-water experiments, the initial gas loading pressure used is based on the amount of gas hydrate desired. Methane hydrate forms from the available free gas in the pore space. At the end of hydrate formation, it is assumed that all remaining free gas is consumed. Since there is an abundant or excess supply of water, it is assumed that one mole of methane gas produces one mole of hydrate. The calculation method for n number of moles of methane gas required for a given hydrate saturation, H_c (%) is taken from Priest (2009) where V_v is the pore volume (m^3), ρ_H is the density of the hydrate (910 kg/m3 from Sloan, 1998), and M_H is the molar mass of the methane hydrate (0.1196 kg/mol from Sloan, 1998).

$$n = \frac{V_v H_c \rho_H}{100 M_H} \tag{3.1}$$

The experimental volume includes both the pore volume and the volume within theblines and pressure transducer. Knowing the exact experimental volume is important since the amount of dissolved methane gas is a function of volume along with salinity (44 ppt KI), pressure, and temperature. Methane solubility is calculated according to various empirical equations (Duan et al., 1992; Duan et al. 2006; Wong, 2005). The number of dissolved moles of methane is combined with the moles calculated from **Eq. 3.1** and used in an equation of state to estimate the initial gas loading pressure (Duan, 1992). Details of these calculations are provided in MATLAB code, Appendix B.2. Code was originally written by Tiannong 'Skyler' Dong.

3.2.2d Electronics for Automated System

1) Elegoo Mega 2560 R3

The Arduino Mega is a larger and more powerful microcontroller board suited for more complex projects. A breadboard schematic of the Mega is shown in **Fig. 3.7**. It has a 16 MHz quartz crystal, 16 analog inputs, 54 digital input/output pins (15 of which can be used as PWM outputs), a USB connection port, a power jack, an ICSP header and a reset button. Power is supplied through either the USB connection port or the power jack. A separate 9V power supply is used to power the Arduino Uno during X-ray scanning. The pins operate at 5V with each pin capable of providing and receiving a maximum of 40 mA.

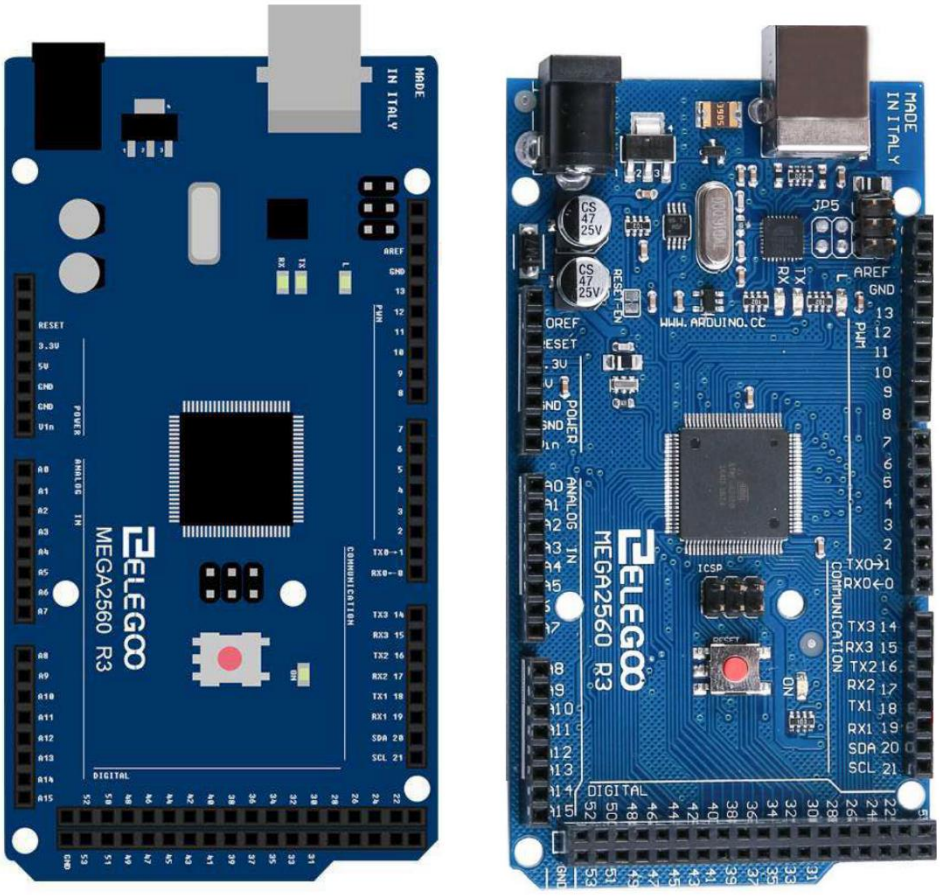

Fig. 3.7—Fritzing breadboard schematic (left) and picture (right) of Elegoo Mega 2560.

2) Adafruit Micro-SD Breakout Board (ADA254)

Since the Arduino microcontroller has limited memory built-in storage, a micro-SD breakout broad is used to log pressure and temperature data. The logic level of the breakout board is strictly 3.3V, however the onboard voltage regulator permits usage with 5V systems. Between the two modes to interface with SD cards, SPI mode was chosen over SDIO mode since the former is easier for any microcontroller to communicate with as only 4 pins are required. The pin connections for SPI mode varies between the Arduino Uno and Arduino Mega 2560 variant.

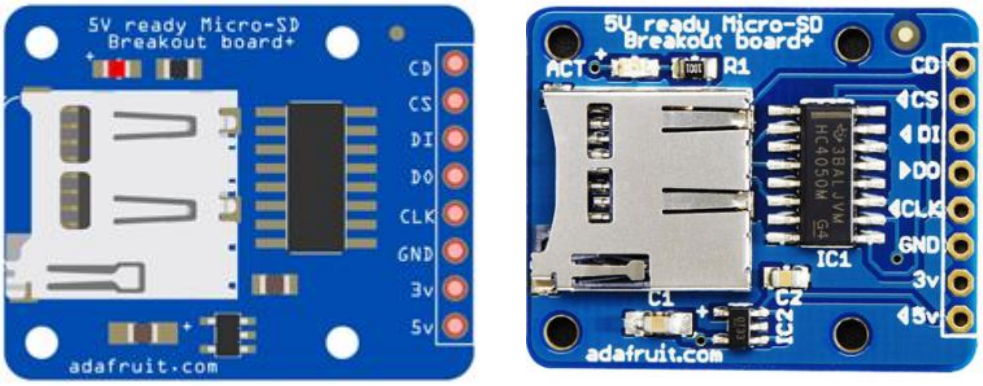

Fig. 3.8—Fritzing breadboard schematic (left) and picture (right) of micro-SD board.

3) DS 3231 Real Time Clock Module

Since the built-in Arduino timekeeper is unreliable due to frequent power cycling of the microcontroller, a separate real time clock (RTC) module is used. Even if power for the Arduino is disconnected, the RTC module continues to keep track of time as it is powered separately. The DS 3231 RTC module shown in **Fig. 3.9** has a temperature compensated crystal oscillator for maintaining I2C real-time accuracy. It has 32K bits of memory organized as 4K by 8 bit memory. The module is powered through a separate 3V lithium battery (LIR2032) and has an operating voltage from 3.3V to 5V. Only four connections are needed: VCC and GND for powering the module and two I2C communication pins, SDA and SCL.

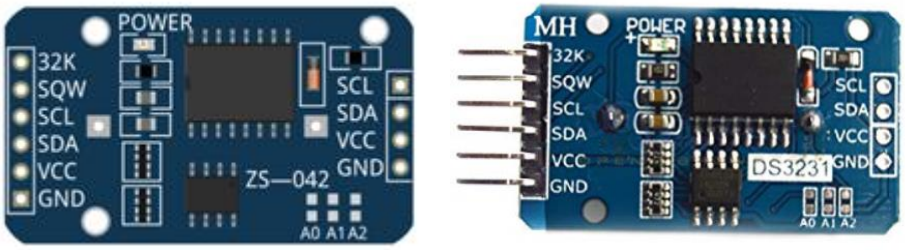

Fig. 3.9—Fritzing breadboard schematic (left) and picture (right) of real time clock.

4) Peltier Cell

Thermoelectric cooling of the sediment bearing vessel to temperatures as low as 1 °C as accomplished using TEC-12706 Peltier cells (**Fig. 3.10**). The cells act as an active heat pump, transferring heat from one side of the cell to the other side through consumption of electrical energy. The cold side of the Peltier cells are oriented towards the ends of the vessel. The hot side of the Peltier cells are oriented away from the vessel where it is attached to a heat sink and fan. If heat isn't properly dissipated from this side, the less effective the cell becomes since the maximum temperature difference between the hot and cold side of a TEC-12706 Peltier element is 65 °C. The amount of heat generated is proportional to current and time (Sharma 2014). Current draw with supply voltage of ~12V is approximately 30A. As long as sufficient power is supplied, Peltier cell performance remains mostly consistent with time (**Fig. 3.11**)

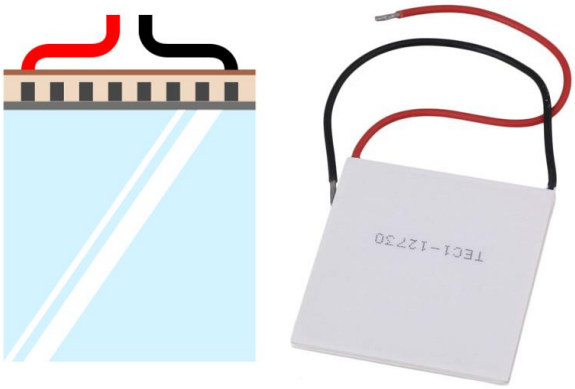

Fig. 3.10—Fritzing breadboard schematic (left) and picture (right) of Peltier cell.

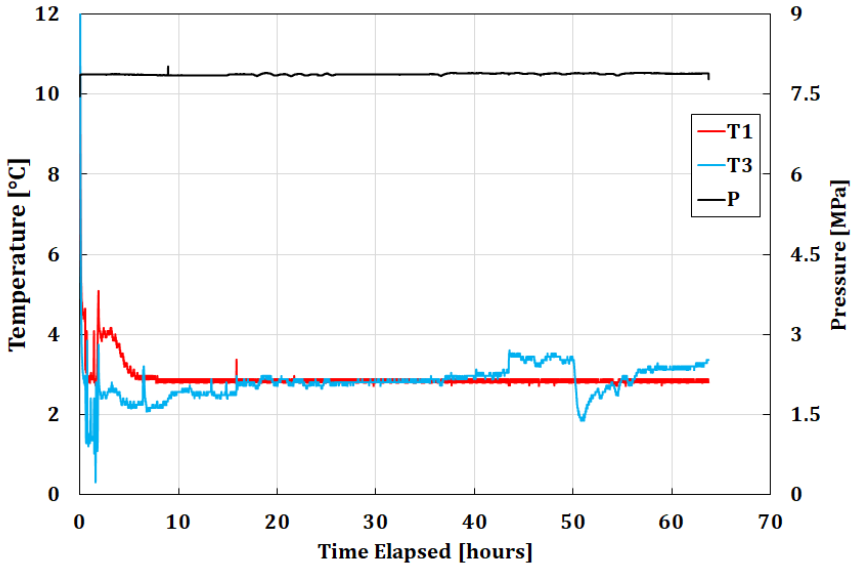

Fig. 3.11—Thermoelectric cooling using Peltier cells stable at ~3 °C for more than 60 hours.

5) 12V Brushless DC Cooling Fan

These fans are directly attached to the aluminum heatsink. Heat from the hot side of the Peltier cell is transferred to the heatsink and dissipated with the cooling fan. Voltage of 12V is directly supplied to these fans for operation

Fig. 3.12—Fritzing breadboard schematic (left) and picture (right) of brushless DC cooling fan.

5) MDD10A Motor Driver

While the Peltier elements can be operated directly by connecting its two terminals to a 12V power source, there is no way to control the rate of cooling. That is why a MDD10A motor driver is used to independently drive both the top and bottom Peltier elements at high currents up to 10A continuously. The driver module has 2 separate PWN ports and DIR ports. Using sign-magnitude pulse width modulation (PWM), the motor driver can directly modulate the duty cycle. The longer the duty cycle, the longer the Peltier element is on and the more power transmitted to the load (Chan, 2018). The motor driver is operated under sign-magnitude mode. In this mode, 2 separate signals are used to control the Peltier element (speed and direction specified individually). For example, for cooling DIR1 and PWN2 are set to HIGH and DIR2 and PWN1 are set to LOW. For heating DIR2 and PWN1 are set to HIGH and DIR1 and PWN2 are set to LOW.

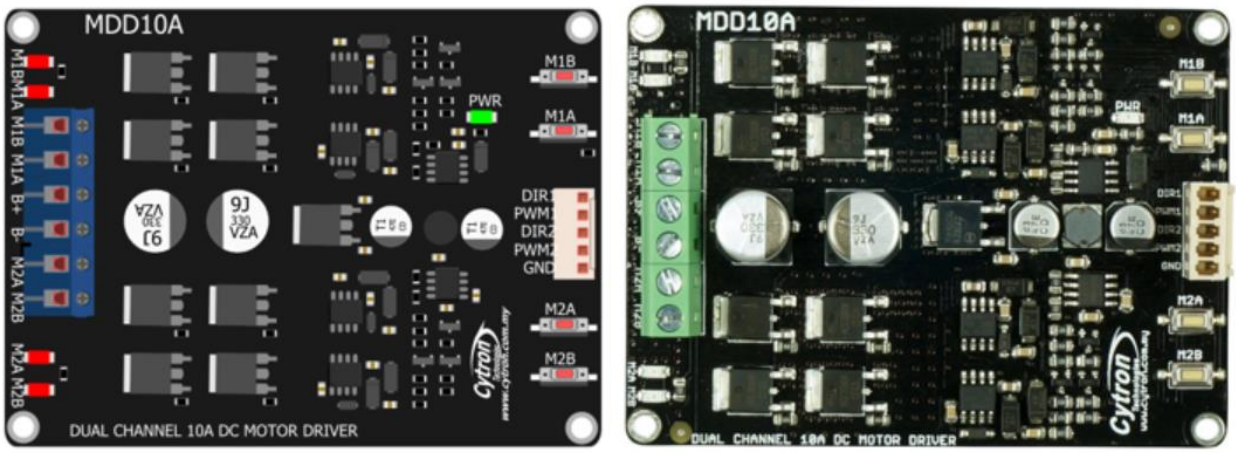

Fig. 3.13—Fritzing breadboard schematic (left) and picture (right) of MDD10A dual motor driver

6) LCD 1602 Module

This alphanumeric liquid crystal display is used to display pressure and temperature readings. This is useful since one can directly monitor the conditions of the experiment without having to remove the micro-SD card to read it. A LED backlight lights up two rows with up to 16 characters in each row. A 10K potentiometer is used to control the contrast and brightness of the display. The LCD module is powered through 5V which can be provided through the Arduino.

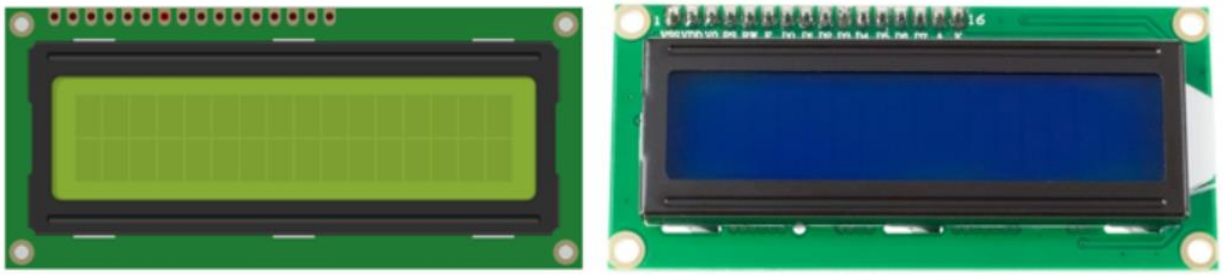

Fig. 3.14—Fritzing breadboard schematic (left) and picture (right) of LCD

7) Thermistor

Negative temperature coefficient (NTC) thermistors are used for temperature measurement. Thermistors are thermal resistors that change its resistance with temperature. In the case of NTC thermistors, resistance increases as temperature decreases. Since the thermistors are read as analog inputs into the microcontroller, the input from the thermistor must first be converted into a resistance using **Eq. 3.2**. Temperature is calculated from the resistance using the Steinhart-Hart equation (**Eq. 3.3**) where $a = 319.72, \mathrm{b} = 36.471, c = 0.052763$ are Steinhart-Hart coefficients specific to the type of thermistor and temperature range of interest. The error associated with the calculation is less than 0.02 °C.

$$R_1 = 19920 \cdot \frac{1023 - A_1}{A_1} \tag{3.2}$$

$$T1 = \mathrm{a} - \mathrm{b}\log(R_1) + \mathrm{c}\log(R_1)^3 \tag{3.3}$$

3.2.2e Arduino IDE for PID Thermoelectric Cooling

The Arduino Integrated Development Environment (IDE) is a software coding tool that allows users to connect and communicate to the Arduino hardware through uploaded sketches. digitalWrite() and digitalRead() are all examples of functions that are used within the IDE environment to control devices connected to any of the Arduino input/output pins. For example, digitalWrite(5,HIGH) triggers a HIGH reading to whatever is connected to pin 5.

The original sketch for PID thermoelectric cooling and data logging was originally written by Dr. Xiongyu Chen (2018). The sketch since then has been modified and is shown in Appendix A.3. The sketch is divided into three sections. The first section specifies all relevant variables and libraries. This includes variables like 'log_period' for specifying the logging period length in microseconds and variables like 'P_value1', which is a proportional PID parameter used to control the top Peltier element's rate of cooling. This section also includes relevant libraries like <SD.h>, <SPI.h>, and <DS3231.h> for data logging and <LiquidCrystal.h> for LCD display. Libraries provide extra functionality for use in sketches and many come preinstalled with the IDE.

The setup section follows next. This section is called once when the sketch is first read. It is used to initialize variables, pin modes, libraries that were defined in the previous section. In this section, the baud rate or serial communication rate is set to 9600. The pin modes specified in the previous section are defined as outputs and their initial values set. For example, in this section the top Peltier element which are controlled using PWM1 and DIR1 of the MDD10A motor driver are initialized and set as outputs. Similarly, PWM2 and DIR2 of the bottom Peltier element are also initialized as outputs. For first time use, the initial time from the DS 3231 real time clock module can be set. This is accomplished through the functions rtc.setTime(16, 22, 0) and rtc.setDate(8, 8,

2017), which sets the time to 4:22 pm, and the date to August 8, 2017, respectively. The LCD screen and the micro-SD card are also initialized in this section for data monitoring and logging.

The loop section is the last part of the code. This part of the code loops continuously, allowing it to actively control the Arduino. This section is further divided into two parts, one for PID control of thermoelectric cooling via the Peltier effect and the other for data logging. For PID control, the rate of cooling for each Peltier element is proportionally controlled by the offset between target temperature and actual temperature from the thermistor reading. The greater the offset, the greater the adjustment to the Peltier output. Thermistor T1 is used to adjust for the top Peltier element. Thermistor T2 is used to adjust for the bottom Peltier element. Thermistor T3 is used for monitoring of bottom vessel temperatures only.

For data logging, time is directly called from the DS 3231 real time clock. Raw pressure readings are converted from analog into actual pressure values and averaged over the logging period of ~22 seconds. Raw temperature readings are first converted into a resistance then temperature calculated from the resistance using the Steinhart-Hart equation. The time, pressure, and temperature values are logged onto a SD card file named ‘myFile’.

3.2.3 X-Ray System

X-ray micro-computed tomography (CT) imaging is a nondestructive imaging technique that gives 3D images of the solid interiors based on X-ray attenuation. CT imaging is performed with a Nikon XTH-225 scanner and CT dataset reconstruction is done using CTPro3D, a software developed by Nikon. The default 225 kV X-ray source is equipped with a reflection target offering a 3 μm spot size. For this project, each μCT scan is a reconstruction of 2,984 projections completed in 25 minutes. Each projection is an average of 2 radiographs with an energy of 125 KV and 135 μA, and an exposure time of 250 ms without any filters. The 16-bit detector captures grayscale CT numbers ranging from 0 to 65535. The resolution of the experiments range from 20-30 μm/voxel. μCT image segmentation is capable of deriving phase saturations within ±2% saturation units.

3.2.4 Image Processing and Analysis

A CT number to brine salinity calibration curve is created from scans of four sandpacks saturated with methane gas and KI brine at 1 atm and different salinities (Chen et al. 2018a). KI brine is used since KI are strong X-ray attenuating salts, 1–2 orders of magnitude stronger than sand and water (Gerward, 1993). The CT images are linearly adjusted to make CTsand = 23500 and CTgas = 5000. The μCT mages and associated histograms (**Fig. 3.15**) show that sand and brine are easily distinguishable except for 9 wt% KI brine.

The calibration results in linear equations between CT number and brine salinity, which allows for the quantification of salinity evolution based on CT number. The linear equation for CT number versus salinity (wt %) for KI brine is:

$$CT\ number = 1972 \cdot salinity + 7445 \tag{3.4}$$

The standard deviation of the actual data from the predicted values is 2.0 wt%. The calculated salinities from CT images are within 2.0 wt% from the actual salinities used (**Fig 3.16**). Since the uncertainty in salinity determination from CT number is ±2.0 wt%, an average KI salinity of 2.0 wt% or less indicates almost pure methane hydrate (Chen et al. 2018a)

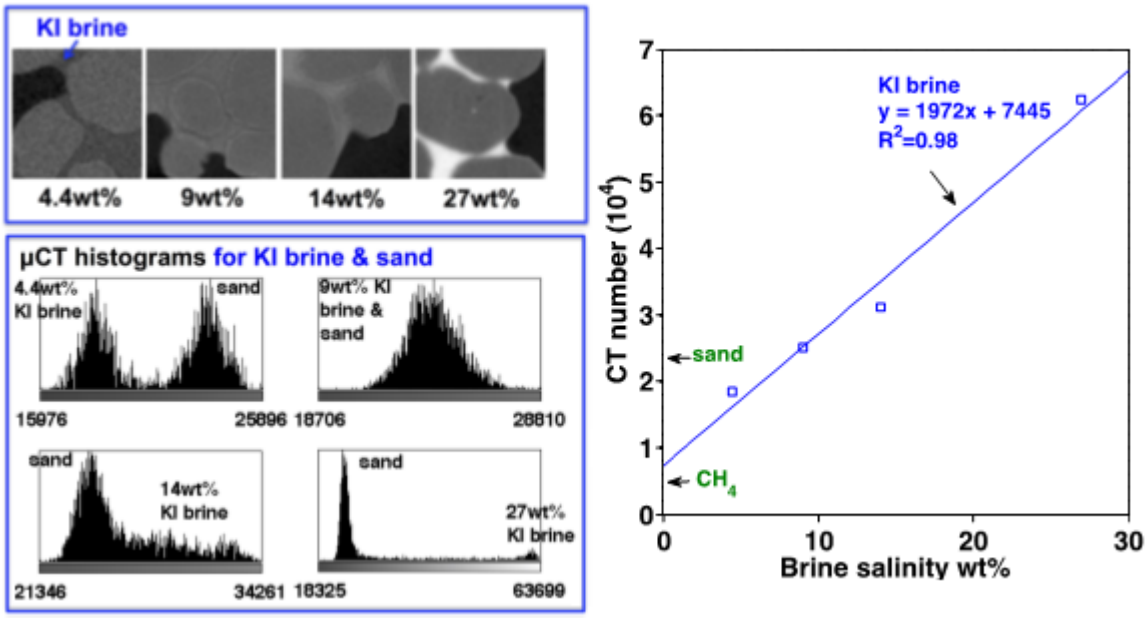

Fig. 3.15—µCT images of sand saturated with methane gas and KI brine at known salinities (top left) and their respective histograms (bottom left). CT number after contrast adjustment versus salinity for KI brine and corresponding linear fit (right).

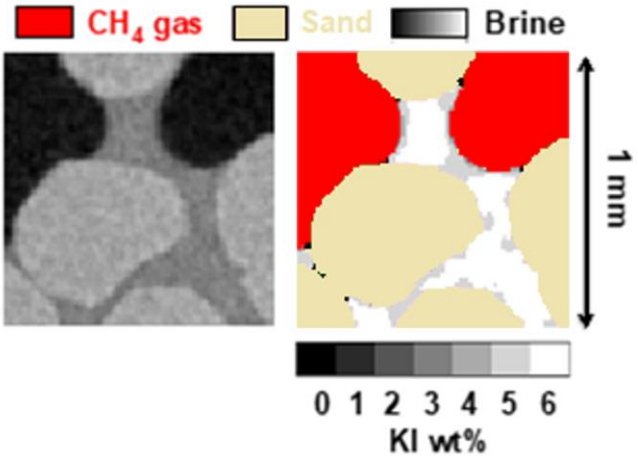

Fig. 3.16—Original (left) and segmented (right) µCT images of sand saturated with methane gas and 4.4 wt% KI brine (Chen et al. 2018a). Both are cropped µCT images of conditions outside of methane hydrate stability.

3.3 Results and Discussion

3.3.1 Overview

The two excess-water experiments conducted for this study follow different pressure-temperature paths (**Fig. 3.17**) into the methane hydrate stability zone. A summary of initial and final conditions of the two experiments are shown in **Tables 3.1** and **3.2**. In the first excess-water experiment, sand is packed at 36% porosity and starts with an initial brine saturation of 20% Methane gas is initially loaded at 22.4°C and 1.59 MPa. The temperature is decreased to 3.0°C and pressure brought up to 8.9 MPa to induce hydrate formation. A time lapse of the 5 day experiment is shown in **Fig. 3.18**.

In the second excess-water experiment, sand is packed at 38% porosity and starts with an initial brine saturation of 21%. Methane gas is initially loaded at 23°C and 0.79 MPa. The temperature is decreased to 2.5°C and pressure brought up to 8.9 MPa to induce hydrate formation. After 4 days, the pressure-temperature conditions for the experiment surpass the methane hydrate stability line for 30 wt% KI brine. A time lapse of the 18 day experiment is shown in **Fig. 3.19**.

Table 3.1—Summary of initial measurements for hydrate experiments

Experiments	Porosity [%]	Brine Saturation [%]	Brine Salinity [wt%]
#1	36	20	4.4
#2	38	21	4.4

Table 3.2—Summary of final measurements for hydrate experiments

Experiments	Elapsed Time [days]	Pressure [MPa]	Temperature [ºC]	Brine Saturation [%]	Hydrate Saturation [%]
#1	5	8.9	3.2	0.9	20
#2	18	8.9	3.3	1.1	25

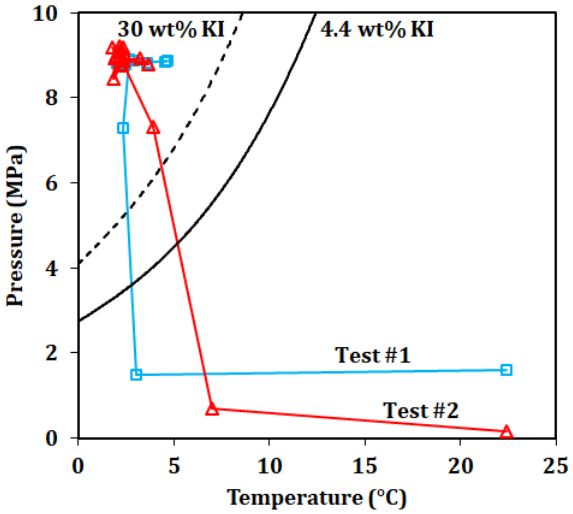

Fig. 3.17—Pressure-temperature path for excess-water experiments #1 and #2. Methane hydrate stability curves are given for 4.4 wt% and 30 wt% KI brine (Tishchenko et al. 2005).

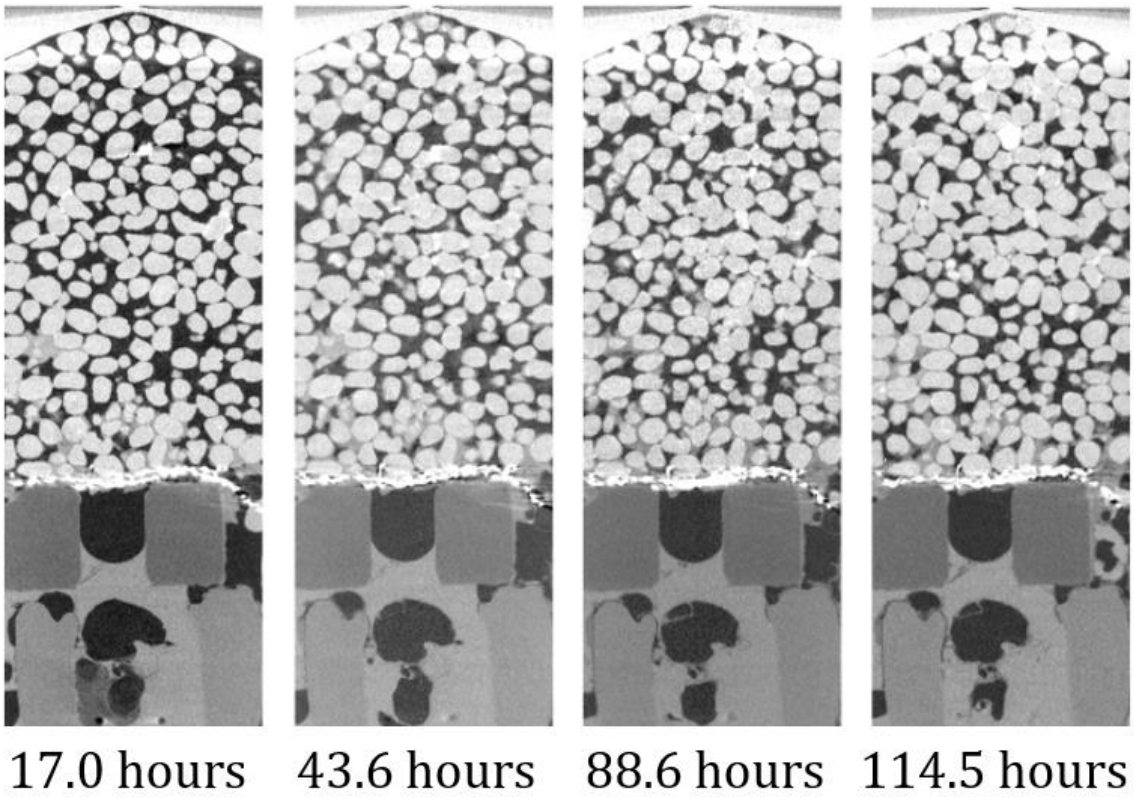

Fig. 3.18—YZ orthoslices of µCT images taken for excess-water hydrate experiment #1

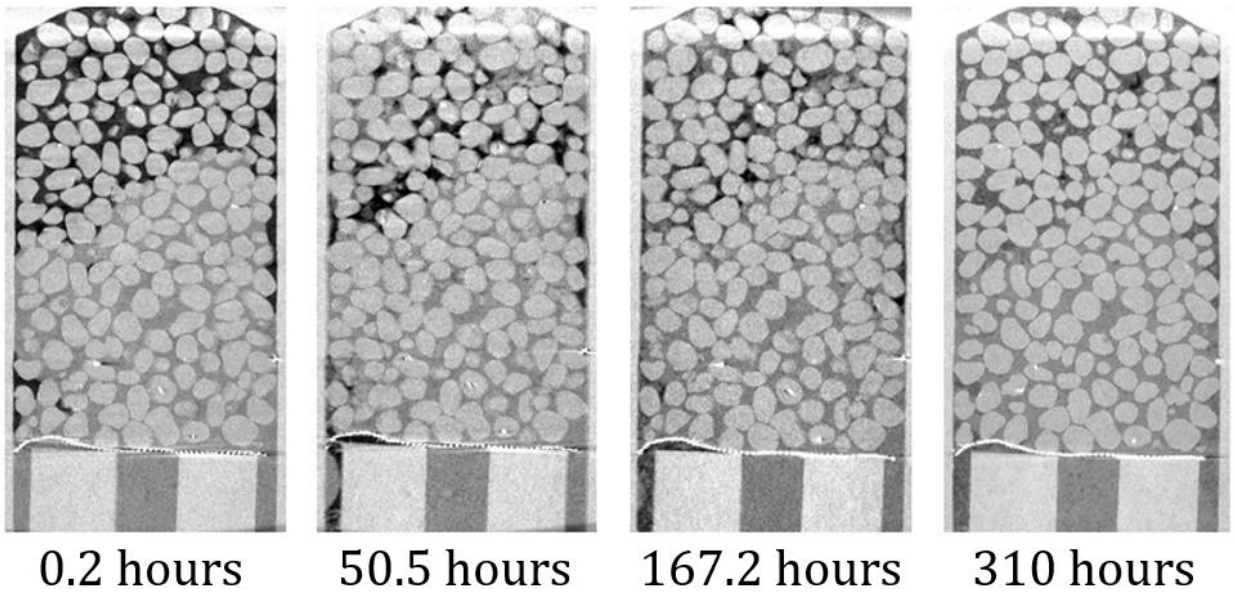

Fig. 3.19—YZ orthoslices of μCT images taken for excess-water hydrate experiment #2

The micro-CT images reveal the hydrate-bearing sand-pack structure. The images show: sand and vessel (light gray), hydrate (dark gray), and methane gas (black). Water CT number varies with salinity, from white (concentrated brine) to light gray (fresh water). Most of the methane hydrate accumulates in the pore space near the top of the vessel. Isolated menisci of concentrated brine are observed throughout the sediment pack at grain contacts. Even though excess-water experiment #2 had a lower initial gas loading pressure, a higher hydrate saturation is observed at the end of the experiment. This can be explained by the duration of the second experiment as it was much longer than the first experiment. The longer the experiment is kept in the methane hydrate stability zone, the more complete the conversion of gas and brine to methane hydrate.

3.3.2 Water-to-Hydrate Conversion Ratio

The water-to-hydrate conversion ratio can be calculated from initial brine saturations and final hydrate saturations. According to water mass balance, dividing the water volume in hydrate by the original available water in the aqueous solution will yield the water-to-hydrate conversion ratio (Chen et al. 2018a):

$$\text{Conversion ratio} = \frac{\rho_{\text{hyd}} \cdot S_{\text{hyd}} \times 5.75 \times 18/(5.75 \times 18 + 16)}{\rho_{bri} \cdot S_{\text{wi}} \cdot (1 - Sal_i)} \tag{3.5}$$

Application of **Eq. 3.5** for excess-water experiment #1 and #2 gives a conversion ratio of 79% and 94%, respectively, assuming hydrate structures of CH4 · 5.75H2O, hydrate density of ρ_{hyd} = 0.9 g/cm3, brine density ρ_{bri} = 1.03 g/cm3 for 4.4 wt% KI brine, and initial brine salinity Sal_i. Compared to the theoretical conversion ratios at three-phase equilibrium derived from water salinity, the conversion ratios from mass balance are slightly higher for experiment #2.

The theoretical conversation ratio according to three-phase equilibrium is the ratio between the water available for hydrate formation (original available water minus water in brine at equilibrium) divided by the original available water (Chen et al. 2018a):

$$\text{Conversion ratio} = \frac{M_{\text{bri}}(1 - \text{Sal}_{\text{i}}) - M_{\text{brf}}\left(1 - \text{Sal}_{\text{eq}}\right)}{M_{\text{bri}}(1 - \text{Sal}_{\text{i}})} = 1 - \frac{\text{Sal}_{\text{i}}}{\text{Sal}_{\text{eq}}} \frac{\left(1 - \text{Sal}_{\text{eq}}\right)}{(1 - \text{Sal}_{\text{i}})} \tag{3.5}$$

Where Sal_{i} is the initial salinity, Sal_{eq} is the salinity at three-phase equilibrium, M_{bri} is the initial mass of brine. Note $M_{\text{bri}} = M_{\text{bri}}(\text{Sal}_{\text{i}}/\text{Sal}_{\text{eq}})$ is the final mass of the brine. **Eq. 3.5** results in a theoretical conversion ratio of 0.89±0.01, taking into account pressure and temperature variability. The higher ratios calculated from the μCT images can be explained by hydrate microporosity, which are small voids that cannot be properly segmented at the current resolution of 11μm (Chen et al. 2018a). This would result in an incorrect hydrate density value in **Eq. 3.4**.

3.3.3 Three-Phase Brine-Gas-Hydrate

Chen (2019c) is able to show the presence of three-phase equilibrium for excess-gas experiments using 1.5 wt% NaBr brine (**Fig. 3.20**). This is also evident with excess-gas and excess-water experiments using 4.4 wt% KI brine. Chen offered several reasons for the presence of three-phase equilibrium and the lack thereof. First, the duration of the experiment plays an important role in three-phase equilibrium. Conditions must stay in the hydrate stability zone long enough to ensure adequate brine conversion to hydrate. Second, the limiting resolution of the μCT scans can prevent identification of non-distinguishable hydrate in the brine or lead to smearing of CT number of thin brine menisci near gas regions resulting in overestimation of hydrate saturation.

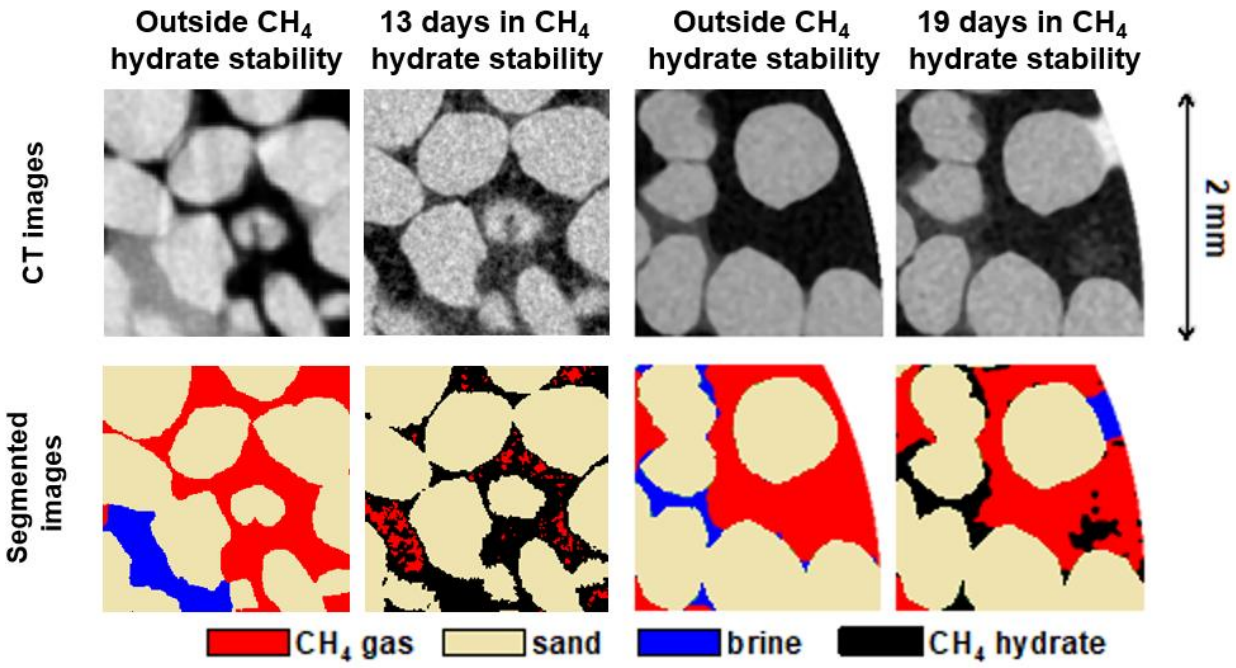

Fig. 3.20—μCT and segmented images for excess-water KI experiment #2 (left) and excess-gas NaBr experiment (right) from Chen et al. (2018a).

3.3.4 Gas and Water Mobility During Hydrate Growth

Based on water droplet experiments (Englezos et al. 1987; Jung et al. 2010; Chen and Espinoza 2018b), a number of hydrate growth models were developed to show that hydrate growth resulted from gas diffusion of the bulk gas phase through solid hydrate into the liquid water phase. The disintegration of hydrate films can also facilitate gas feed for further hydrate growth (Meyer 2018). Our results from excess-water experiments, along with previous results from Chen's excess-gas experiments (2018a) show that a highly mobile water phase also contributes greatly to hydrate formation.

Both Chen's (2018a) excess-gas experiments and our excess-water experiments show that hydrate preferentially form at the water-gas interface. This can be explained by a highly mobile water phase since hydrate formation can only be sustained at locations where brine was originally absent if the brine in the pore space of the sand was mobile enough over long distances. We show evidence of continuous changes of water salinity over distances of tens of grains even at residual brine saturation, hydrate ripening in which preferential growth of hydrate occurs in areas of existing hydrate saturation, and replacement of brine by gas in porous hydrate, which makes hydrate porosity evident (**Figs. 3.21-3.24**).

One possible mechanism for water mobility is flow driven by capillarity through conduits made of mineral surfaces (Chen et al. 2018a). We observe short-lived hydrate menisci coated with brine bridging across grains in both excess-gas and excess-water experiments (**Fig. 3.25**). These 4±2 μm in thickness menisci likely contribute to water transfer and mobilization during hydrate growth. Other possible mechanisms for water mobility could be water vapor condensation, diffusion, and evaporation (Sun et al. 2018).

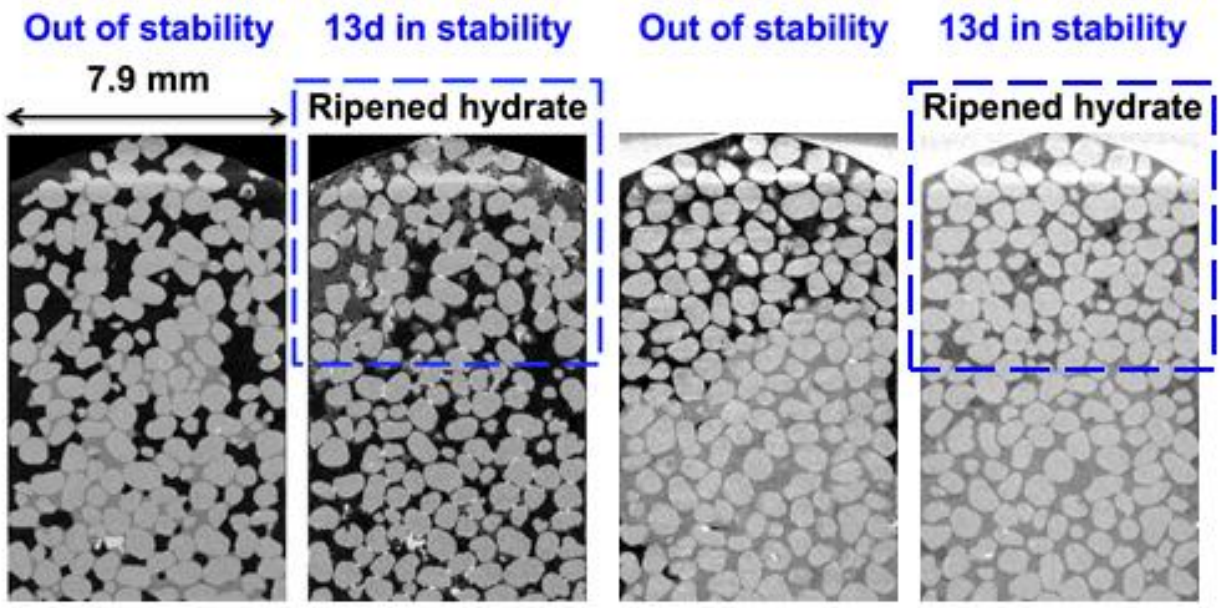

Fig. 3.21—Axial CT slices of excess-gas experiment (left) from Chen et al. (2018a) and excess-water experiment #2 (right) showing conditions out of hydrate stability zone with sand partially saturated with brine/methane and conditions within stability zone with sand saturated with aggregated, interconnected hydrate.

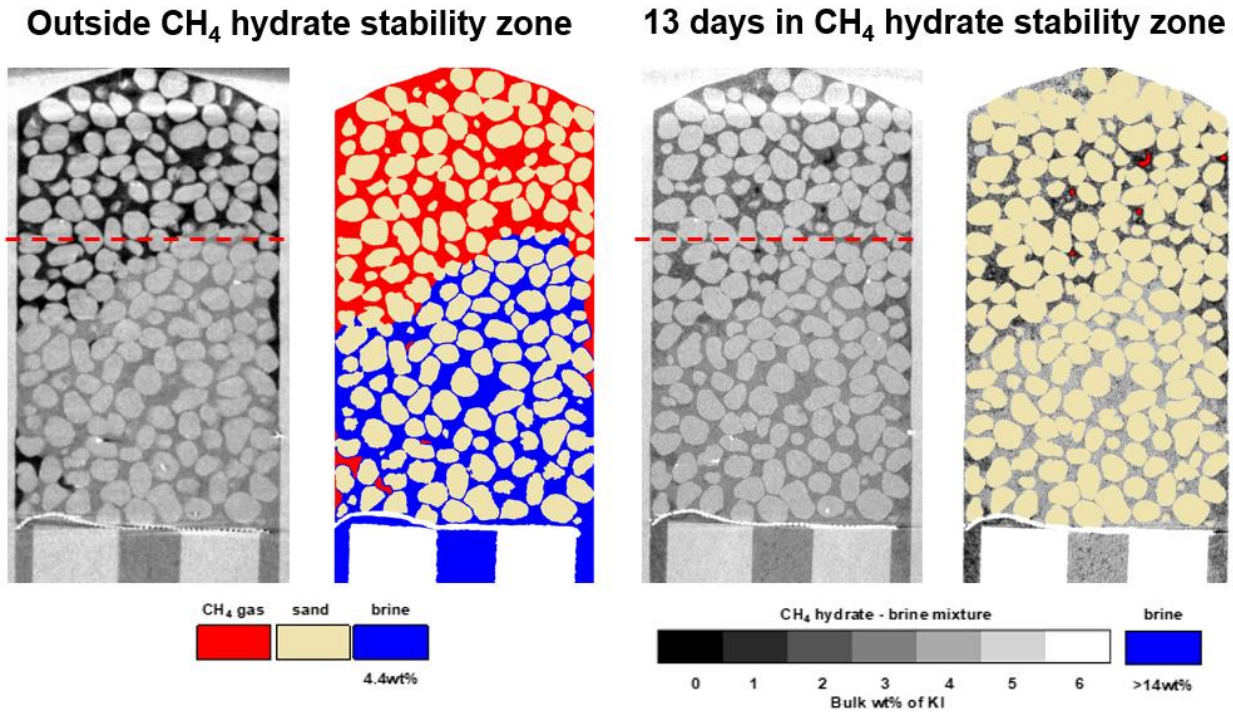

Fig. 3.22—YZ orthoslice µCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2. The images show pore-filling hydrate mixed with brine that evolve into separate porous hydrate and high salinity brine phases. Axial µCT and segmented slices of this experiment (dashed red line) shown in Fig. 3.23.

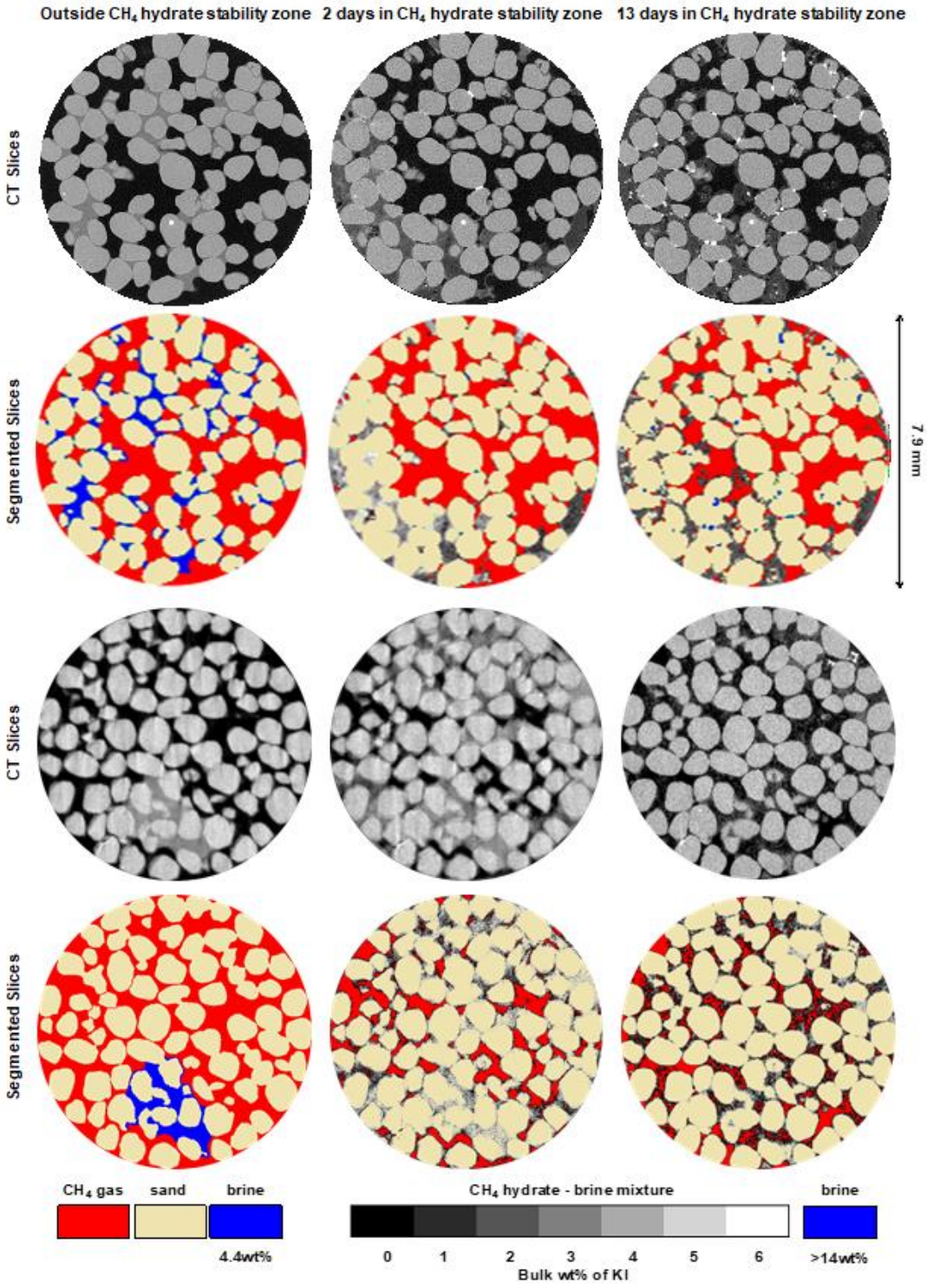

Fig. 3.23—Axial µCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-gas experiment #2 (top) from Chen et al. (2018a) and excess-water experiment #2 (bottom). The images show pore-filling hydrate mixed with brine that evolve into separate porous hydrate and high salinity brine phases

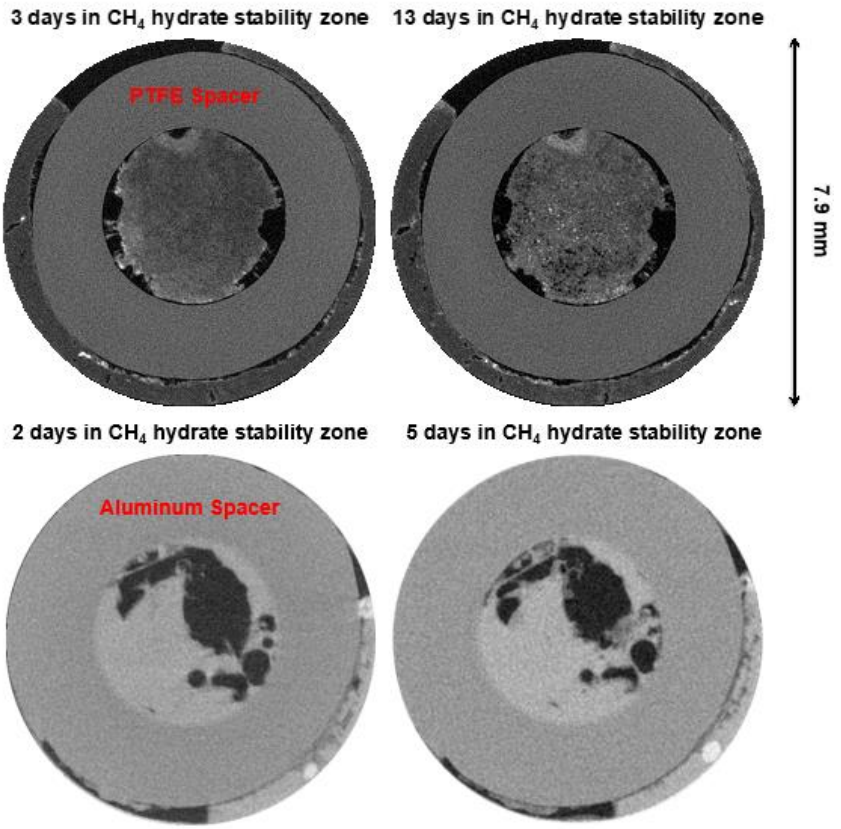

Fig. 3.24—Axial µCT slices of excess-gas experiment (top) from Chen et al. (2018a) showing more aggregated hydrate-rich phase within the bottom PTFE spacer, which did not contain water initially, than excess-water experiment #1 (bottom). As more hydrate grows, more high-salinity brine (white) appears within the hydrate aggregate and the porosity within hydrate becomes apparent (black).

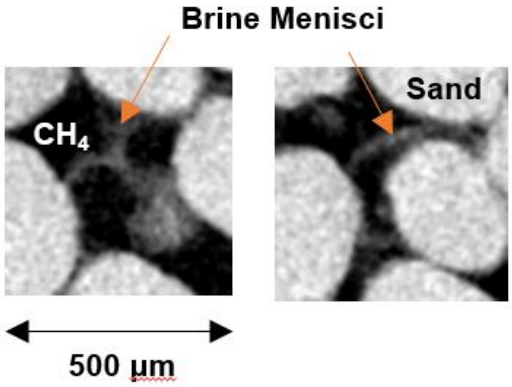

Fig. 3.25—Short-lived brine menisci across grains during methane hydrate growth of excess-water experiment #1.

3.3.5 Hydrate Pore-Habit and Micro-Morphology

Both excess-water and excess-gas experiments show evolving hydrate pore habit during experimental lengths of as long as 2 weeks. Hydrate preferentially forms in interconnected pore bodies and exhibits a patchy, heterogeneous distribution with hydrate saturation significantly exceeding the original bulk brine saturation in some locations. This is a result of the movement of both gas and water, which ultimately results in a heterogeneous hydrate distribution – over 50% of the total hydrate mass residing in 20% of the pore space. Chen refers to this process of hydrate aggregation as ripening, an analog to Ostwald ripening in which preferential growth of hydrate occurs in areas of existing hydrate saturation.

These observations show that the evolution of hydrate pore habit with time should be taken into account, especially in laboratory measurements done over a few days as enough time should be given to ensure adequate mobilization and conversion of brine and gas to methane hydrate. A good benchmark for adequate conversion of brine and gas to methane hydrate during excess-gas experiments is pressure. Ideally, pressure will gradually decrease as saturation of hydrate increases, but will level off as finite water is consumed. In excess-water experiments, hydrate saturation is restricted by the amount of added gas. Ideally, water intake will gradually increase as saturation of hydrate increases, but level off as finite methane is consumed.

3.3.6 Gas Hydrate Dissociation

Chen and Espinoza (2018c) were able to integrate X-ray radiography and μCT to measure xenon hydrate dissociation kinetics in a porous sand medium. He found that [hydrate volume]$^{2/3}$ or surface area is mostly a linear metric of hydrate dissociation rate except in two scenarios. The first scenario occurs when critical hydrate saturation goes above 35–45%. In this scenario, hydrate dissociation rate initially stays constant and then decreases with [hydrate volume]$^{2/3}$. In the second scenario, the dissociation rates of patchy hydrates in large pores stay relatively constant and are significantly slower than the dissociation rates of hydrates in small pores.

In this project, we use μCT instead of X-ray radiography to directly observe methane hydrate dissociation, which allows us to better visualize the evolution of the pore habit during this process. After running excess-water experiment #2 for 18 days, pressure is lowered to 400 psi to bring conditions outside of the hydrate stability zone (**Fig. 3.26**). Dissociation is continued for 43 hours at constant pressure and temperature conditions.

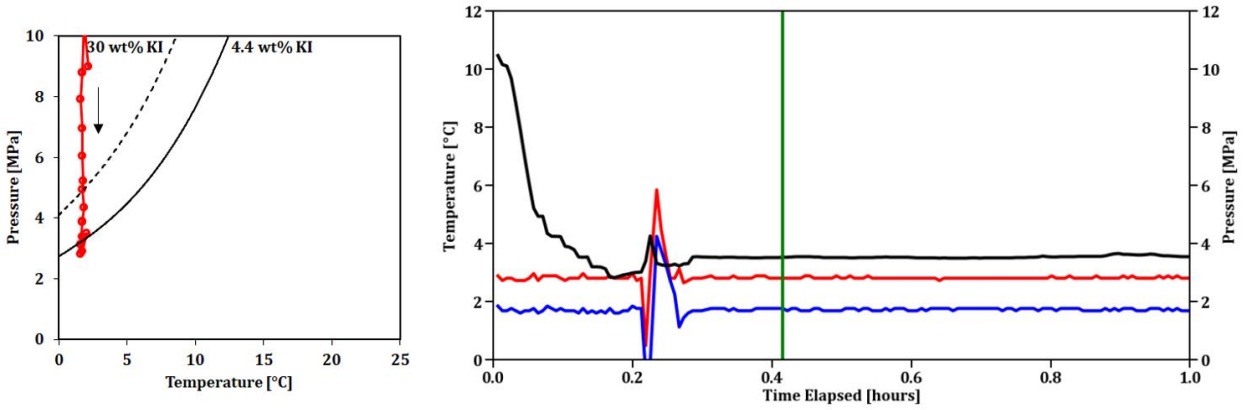

Fig. 3.26— Pressure-temperature path (left) and pressure decay curve for dissociation experiment. Dissociation started 18.2 days into excess-gas experiment #2.

μCT images reveal the evolution of the hydrate-bearing structure during dissociation (**Figs. 3.27** and **3.28**). Before dissociation, methane hydrate accumulates in the pore space near the top of the vessel. Upon dissociation, hydrate mass decreases. Hydrate located near large gas-filled pores and the vessel walls dissociate first. Hydrate preferentially dissociates next to the vessel walls because of latent heat transfer along the vessel walls. During dissociation, the water from dissociated hydrate mixes with concentrated brine and forms a connected brine phase with gas pockets. The images show a very distinct hydrate habit after dissociation.

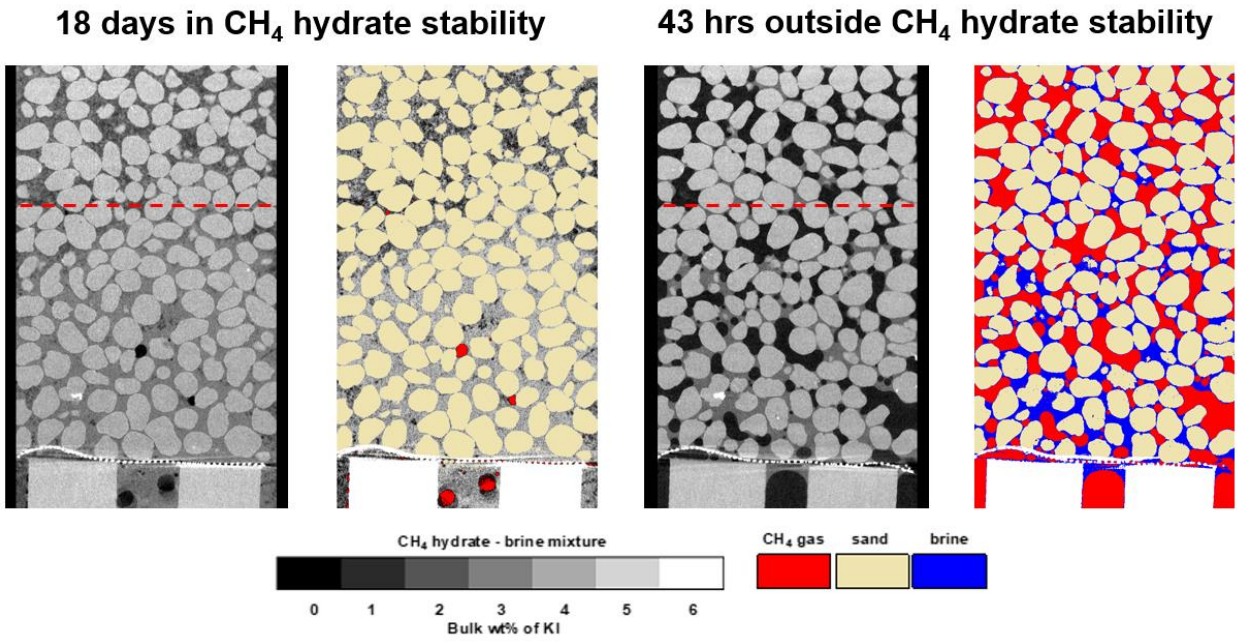

Fig. 3.27—YZ orthoslice μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2 during dissociation. Axial μCT and segmented slices of this experiment (dashed red line) shown in Fig. 3.28.

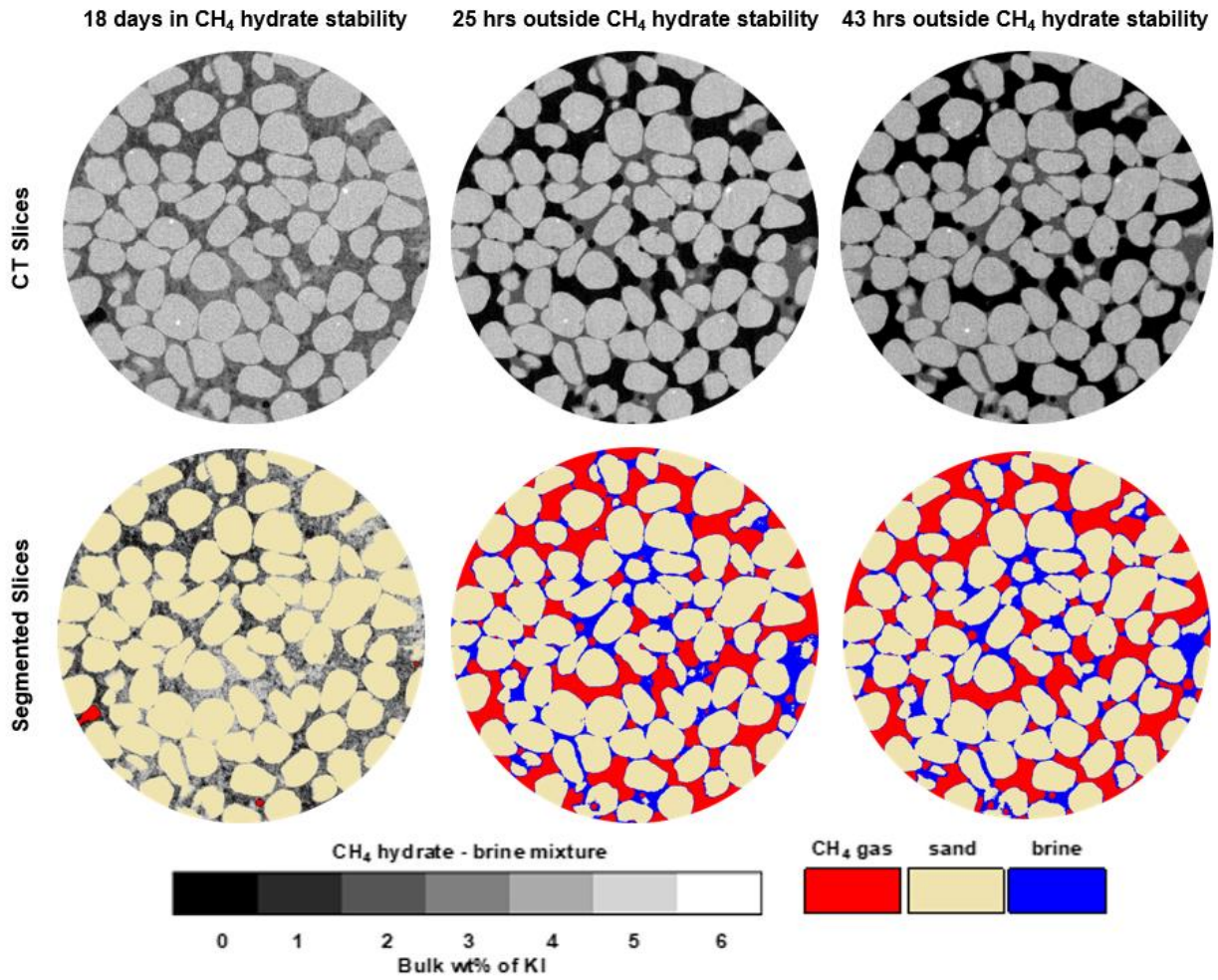

Fig. 3.28—μCT and segmented slices with salinity calculation showing evolution of KI wt% in brine and brine-hydrate mixtures for excess-water experiment #2 during dissociation.

3.4 Conclusions

In this study, we utilize a micro-consolidation vessel to study methane hydrate growth from KI brine within the pore space at both excess-gas and excess-water conditions. Based on our results, the following conclusions can be drawn:

- The calibration of CT number with salinity allows us to quantify ion exclusion within brine-hydrate mixtures and monitor brine salinity increase during hydrate formation.
- Distribution of high salinity brine from ion exclusion affects hydrate micro-morphology during hydrate growth.
- Water is mobile over fairly long distances during hydrate growth and its movement facilitates hydrate ripening in sediments, resulting in interconnected pore habit and distinct hydrate saturations over small scales.
- Hydrate preferentially dissociates near large gas-filled pores and next to the vessel walls because of latent heat transfer along the vessel walls.

Appendices

Appendix A: Arduino IDE Sketches

A.1 Project 1: Flow Automation and Data Logging

```
/* ----- SECTION 1: VARIABLES AND LIBRARIES ----- */

/* -- Pin Definitions -- */
// For valve control
#define Water 5
#define Gas 4
#define Other 3
// For datalogging
#define PinCS 10

/* -- Libraries -- */
// For datalogging
#include <SD.h>
#include <SPI.h>
#include <DS3231.h>

/* -- Initial Conditions -- */
// For valve control
int Received = 0;
int WaterState = 0;
int GasState = 0;
int OtherState = 0;
// For datalogging
File myFile;
DS3231  rtc(SDA, SCL);

/* ----- SECTION 2: SETUP ----- */

void setup()
{
        // Serial setup
        Serial.begin(9600); // default communication rate of the HC-06 Bluetooth module
        // Valve control steup
        pinMode(Water,OUTPUT); // set pins as outputs
        digitalWrite(Water,LOW); // valves off by default
        pinMode(Gas,OUTPUT);
        digitalWrite(Gas,LOW);
        pinMode(Other,OUTPUT);
        digitalWrite(Other,LOW);

        // For datalogging
        pinMode(PinCS, OUTPUT);
        Serial.println("Initializing SD card...");
```

```
	if (SD.begin(PinCS))
	{
		Serial.println("SD card is ready to use.");
	}
	else
	{
		Serial.println("SD card initialization failed");
		return;
	}

	rtc.begin();
}

/* ----- SECTION 3: LOOP ----- */

void WAG()
{
	if(Serial.available() > 0) // Checks whether data is coming from the serial port
  {
		Received = Serial.read(); // "Received" variable is data read from the serial port
  }

	/* -- Water -- */
	// When bluetooth receives '1', meaning WaterOn button was pressed and
	// WaterState is zero, meaning Water valve is closed, this code turns valve on
	if ((WaterState == 0) && (Received == '1'))
  {
		digitalWrite(Water,HIGH);
		Serial.println("Water: ON");
		WaterState = 1;
		Received = 0;
  }
	// When bluetooth receives '2', meaning WaterOff button was pressed and
	// WaterState is zero, meaning Water valve is closed, this code turns valve off
	if ((WaterState == 0) && (Received == '2'))
  {
		digitalWrite(Water,LOW);
		Serial.println("Water: OFF");
		WaterState = 0;
		Received = 0;
  }
	// When bluetooth receives '1', meaning WaterOn button was pressed and
	// WaterState is one, meaning Water valve is open, this code turns valve open
  if ((WaterState == 1) && (Received == '1'))
  {
		digitalWrite(Water,HIGH);
		Serial.println("Water: ON");
		WaterState = 1;
		Received = 0;
  }
```

```
        // When bluetooth receives '2', meaning WaterOff button was pressed and
        // WaterState is one, meaning Water valve is open, this code turns valve off
if ((WaterState == 1) && (Received == '2'))
{
                digitalWrite(Water,LOW);
                Serial.println("Water: OFF");
                WaterState = 0;
                Received = 0;
}
        /* -- Gas -- */
        // When bluetooth receives '3', meaning GasOn button was pressed and
        // GasState is zero, meaning Gas valve is closed, this code turns valve on
if ((GasState == 0) && (Received == '3'))
{
                digitalWrite(Gas,HIGH);
                Serial.println("Gas: ON");
                GasState = 1;
                Received = 0;
}
        // When bluetooth receives '4', meaning GasOff button was pressed and
        // GasState is zero, meaning Gas valve is closed, this code turns valve off
if ((GasState == 0) && (Received == '4'))
{
                digitalWrite(Gas,LOW);
                Serial.println("Gas: OFF");
                GasState = 0;
                Received = 0;
}
        // When bluetooth receives '3', meaning GasOn button was pressed and
        // GasState is one, meaning Gas valve is open, this code turns valve open
if ((GasState == 1) && (Received == '3'))
{
                digitalWrite(Gas,HIGH);
                Serial.println("Gas: ON");
                GasState = 1;
                Received = 0;
}
        // When bluetooth receives '4', meaning GasOff button was pressed and
        // GasState is one, meaning Gas valve is open, this code turns valve off
if ((GasState == 1) && (Received == '4'))
{
                digitalWrite(Gas,LOW);
                Serial.println("Gas: OFF");
                GasState = 0;
                Received = 0;
}

/* -- Other -- */
// When bluetooth receives '5', meaning On button was pressed and
// OtherState is zero, meaning valve is closed, this code turns valve on
```

```
  if ((OtherState == 0) && (Received == '5'))
  {
                        digitalWrite(Other,HIGH);
                        Serial.println("Other: ON");
                        OtherState = 1;
                        Received = 0;
  }
 // When bluetooth receives '6', meaning Off button was pressed and
 // OtherState is zero, meaning valve is closed, this code turns valve off
  if ((OtherState == 0) && (Received == '6'))
  {
                        digitalWrite(Other,LOW);
                        Serial.println("Other: OFF");
                        OtherState = 0;
                        Received = 0;
  }
 // When bluetooth receives '5', meaning On button was pressed and
 // OtherState is one, meaning valve is open, this code turns valve open
  if ((OtherState == 1) && (Received == '5'))
  {
                        digitalWrite(Other,HIGH);
                        Serial.println("Other: ON");
                        OtherState = 1;
                        Received = 0;
  }
 // When bluetooth receives '6', meaning Off button was pressed and
 // OtherState is one, meaning  valve is open, this code turns valve off
  if ((OtherState == 1) && (Received == '6'))
  {
                        digitalWrite(Other,LOW);
                        Serial.println("Other: OFF");
                        OtherState = 0;
                        Received = 0;
  }
}

void DATALOGGING()
{
        int Reading = analogRead(A0); //ranges from 100 to 900 (0 to 200 psig) (0.5V to 4.5V)
        float Voltage = ((Reading - 100) * (0.005)) + 0.5; //(4.5-0.5)/(900-100) = 0.005
        float Pressure = (Reading - 100) * (0.25); //(200-0)/(900-100) = 0.25

        Serial.print(rtc.getTimeStr());
        Serial.print(",");
        Serial.print(int(rtc.getTemp()));
        Serial.print(",");
        Serial.print(Reading);
        Serial.print(",");
        //Serial.print(Voltage);
        //Serial.print(",");
```

```
		Serial.println(Pressure);

		myFile = SD.open("data.txt", FILE_WRITE);
		if (myFile)
		{
			myFile.print(rtc.getTimeStr());
			myFile.print(",");
			myFile.print(int(rtc.getTemp()));
			myFile.print(",");
			myFile.print(Reading);
			myFile.print(",");
			//myFile.print(Voltage);
			//myFile.print(",");
			myFile.println(Pressure);
			myFile.close();
		}
		else
		{
			Serial.println("error opening data.txt");
		}
		delay(1000);
}

void loop()
{
		WAG();
		DATALOGGING();
}
```

A.2 Project 1: Syringe Pump

```
/* ----- SECTION 1: VARIABLES AND LIBRARIES ----- */

/* -- Pin Definitions -- */
const int StepPin = 3;
const int DirPin = 2;

/* -- Initial Conditions -- */
String Received = "";
String Volume = "0 mL";
String Rate = "0 cc/min";
String MotorDirection = "";
String Mode = "";
boolean DirRotation = HIGH;
int MicroSecondDelay = 41.4*0.27;
int StepCounter = 0;

/* -- Constants -- */
#define SYRINGE_VOLUME_ML 53
#define SYRINGE_BARREL_DIAMETER_MM 21.20
#define SYRINGE_BARREL_LENGTH_MM 150
#define THREADED_ROD_PITCH 1.25
#define MICROSTEPS_PER_STEP 16.0
#define STEPS_PER_REVOLUTION 20000.0

// The NEMA 17 stepper motor has 20000 steps per revolution, meaning each step is 0.018 degrees.
// The micro stepping driver allows 16 micro-steps (intermediate) in one stepper motor step.
// The motor is attached to the lead-screw, the lead-screw has a 1.25mm pitch. This means that for each
360 rotation of the lead screw the nut screw will dislocate 1.25mm.
long MicroSteps1MM = MICROSTEPS_PER_STEP * STEPS_PER_REVOLUTION /
THREADED_ROD_PITCH; // 256000 microsteps for 1 mm
long MicroSteps1ML = (MICROSTEPS_PER_STEP * STEPS_PER_REVOLUTION *
SYRINGE_BARREL_LENGTH_MM) / (SYRINGE_VOLUME_ML * THREADED_ROD_PITCH );
// 724,528 microsteps for 1 mL

// For rate calculations:
// If our STEP signal is 1ms high and 1ms low, each complete pulse will take 2ms of time.
// digitalWrite(9, HIGH);
// delay(1);
// digitalWrite(9, LOW);
// delay(1);
// Since there are 1000ms in 1 second, then 1000/2 = 500 microsteps/second.
```

```
/* ----- SECTION 2: SETUP ----- */

void setup()
{
	// Serial Setup
	Serial.begin(9600); // default communication rate of the HC-06 Bluetooth module
	// Motor Setup
	pinMode(StepPin,OUTPUT);
	pinMode(DirPin,OUTPUT);
	digitalWrite(DirPin, DirRotation); //injecting by default
}

/* ----- SECTION 3: LOOP ----- */

void loop()
{
	if(Serial.available() > 0) // Checks whether data is coming from the serial port
	{
		Received = Serial.readString(); // Reads the data from the serial port
	}

	/* Injection Rate */
	if (Received == "1 cc/min")
	{
		Rate = Received;
		MicroSecondDelay = 41.4*0.27; // (1000/(2*(1*MicroSteps1ML/60))) = 0.0414 s
		// 0.27 is a fudge factor since the gearbox efficiency is ~73%
	}
	if (Received == "10 cc/min")
	{
		Rate = Received;
		MicroSecondDelay = 4.14*0.27; // (1000/(2*((10*MicroSteps1ML)/60))) = 0.00414
	}
	if (Received == "Max Speed")
	{
		Rate = Received;
		MicroSecondDelay = 0;
	}

	/* Motor Direction */
	if (Received == "Inject")
	{
		MotorDirection = Received;
	}
	if (Received == "Withdraw")
	{
		MotorDirection = Received;
	}
```

```
		/* Motor Mode */
		if (Received == "Start")
		{
			Mode = Received;
		}
		if (Received == "Stop")
		{
			Mode = Received;
		}

		if (Mode == "Start")
		{
			if (MotorDirection == "Inject")
			{
				DirRotation = HIGH;
				digitalWrite(DirPin,DirRotation);
				MotorDirection = "";
			}
			if (MotorDirection == "Withdraw")
			{
				DirRotation = LOW;
				digitalWrite(DirPin,DirRotation);
				MotorDirection = "";
			}
			digitalWrite(StepPin,HIGH);
			delayMicroseconds(MicroSecondDelay);
			digitalWrite(StepPin,LOW);
			delayMicroseconds(MicroSecondDelay);
			StepCounter = StepCounter + 1;
		}

		if (Mode == "Stop")
		{
			Mode = "";
			Serial.println(StepCounter);
		}
}
```

A.3 Project 3: PID Thermoelectric Cooling and Data Logging

```
// Originally written by Dr. Xiongyu Chen
// Code additions for real time clock, LCD display, and thermistor T3 monitoring made by Jeffery Luo

/* ----- SECTION 1: VARIABLES AND LIBRARIES ----- */

/* -- Libraries -- */
#include <SD.h>
#include <SPI.h>
#include <DS3231.h>
#include <math.h>
#include <LiquidCrystal.h>

/* -- Initial Conditions -- */

int log_period = 2000; // Logging period length = log_period/100 seconds
int maxtime = 10000; // This is the period in microseconds
int count=1;

// For datalogging
const int PinCS = 53;
File myFile;
DS3231  rtc(SDA, SCL);

// For pressure recording
double logp = 0;
double singlep = 0; // unit MPa
double sump = 0; // unit MPa
double actualp = 0; // unit MPa

// For temperature control
double D1=1023; // Voltage at A0, for thermistor 1 (top)
double D2=1023; // Voltage at A1, for thermistor 2 (bottom)
double D3=1023; // Voltage at A1, for thermistor 2 (bottom)
double T1=23; // Temperature measured by thermistor 1
double T2=23; // Temperature measured by thermistor 2
double T3=23; // Temperature measured by thermistor 3
// Initialization of T1 and T2 and T3 at room temperature
double R1=20; // Resistance of thermistor 1
double R2=20; // Resistance of thermistor 2
double R3=20; // Resistance of thermistor 3
double Rv1=19920; // Voltage divider for R1
double Rv2=19920; // Voltage divider for R2
double Rv3=19920; // Voltage divider for R3
double lnR1=20; // Natural log of R1
double lnR2=20; // Natural log of R2
double lnR3=20; // Natural log of R3
```

```
// For PID control
double P_value1=1000; // PID parameters, proportional
double P_value2=1000; // PID parameters, proportional
double I_value=0; // PID parameters, integrative
double T_tar= 2; // Targeted temperature in C
double T_offset1=T1-T_tar; // Temperature offset in C
double T_offset2=T2-T_tar; // Temperature offset in C
double T_cumul; // cumulative T_offset along the time, for I_value

// For motor driver
int pwm1 = 5;
int dir1 = 4;
int pwm2 = 3;
int dir2 = 2;
// pwm1 and dir2 HIGH, dir1 and pwm2 LOW, Device heating
// dir1 and pwm2 HIGH, pwm1 and dir2 LOW, Device cooling
// All low, device idle
double outpercent1=T_offset1*P_value1;
double outpercent2=T_offset2*P_value2;
// -Min voltage of 13.02 V.
// This outpercent is [-100, 100]
// Positive means cooling, negative means heating
double x1;
double x2;
// This is the linear transformation from output percentage to the high duration
int high_duration1;
int high_duration2;
// After testing, this works when outpercent is between 11 and 100, but I think the output should be within
[11, 80]%
// In most ranges, the relative error is 1% of expected voltage.

// For LCD
const int rs = 34, en = 35, d4 = 36, d5 = 37, d6 = 38, d7 = 39;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

/* ----- SECTION 2: SETUP ----- */

void setup()
{
        Serial.begin(9600);
        pinMode(pwm1, OUTPUT);
        pinMode(dir1, OUTPUT);
        pinMode(pwm2, OUTPUT);
        pinMode(dir2, OUTPUT);

        rtc.begin();
        // The following lines can be uncommented to set the date and time
        // rtc.setDOW(MONDAY); // Set Day-of-Week to SUNDAY
        // rtc.setDate(11, 2, 2019); // Set the date to February 2nd, 2019
```

```
	// rtc.setTime(16, 16, 0); // Set the time to 12:00:00 (24hr format)

	// Set up the LCD's number of columns and rows:
	lcd.begin(16, 2);
	lcd.setCursor(0, 0);
	lcd.print("T1T2");
	lcd.setCursor(0, 1);
	lcd.print("T3");
	lcd.setCursor(9, 1);
	lcd.print("P");

	// For datalogging
	pinMode(PinCS,OUTPUT);
	Serial.print("Initializing SD Card... ");
	if (SD.begin(PinCS))
	{
		Serial.println("SD card is ready to use.");
	}
	else
	{
		Serial.println("SD card initialization failed");
		return;
	}
	myFile=SD.open("Data.txt",FILE_WRITE);
	myFile.println(".....");
	myFile.close();
}

/* ----- SECTION 3: LOOP ----- */

void PIDControl()
{
	// Give the manual calibrated output to cool down or heat up
	// positive is cooling, negative is warming
	// 1 is top, 2 is bottom

	// 1 is cooling
	if(outpercent1 > 0)
	{
		if(outpercent1 > 92.4)
		{
			outpercent1 = 92.4;
		}

		// 2 is cooling too
		if(outpercent2 > 0)
		{
			if(outpercent2 > 92.4)
			{
				outpercent2 = 92.4;
```

```
		}

		digitalWrite(dir1, LOW); // current from B to A cooling
		digitalWrite(dir2, LOW); // current from B to A cooling

		x1 = 10824.94*(0.01*(outpercent1))-6.66;
		x2 = 10824.94*(0.01*(outpercent2))-6.66;

		if(x1>x2)
		{
			high_duration1 = x2;
			high_duration2 = x1 - x2;

			digitalWrite(pwm1, HIGH);
			digitalWrite(pwm2, HIGH);
			delayMicroseconds(high_duration1);

			digitalWrite(pwm1, HIGH);
			digitalWrite(pwm2, LOW);
			delayMicroseconds(high_duration2);

			digitalWrite(pwm1, LOW);
			digitalWrite(pwm2, LOW);
			delayMicroseconds(maxtime - x1);
		}

		else
		{
			high_duration1 = x1;
			high_duration2 = x2 - x1;

			digitalWrite(pwm1, HIGH);
			digitalWrite(pwm2, HIGH);
			delayMicroseconds(high_duration1);

			digitalWrite(pwm1, LOW);
			digitalWrite(pwm2, HIGH);
			delayMicroseconds(high_duration2);

			digitalWrite(pwm1, LOW);
			digitalWrite(pwm2, LOW);
			delayMicroseconds(maxtime - x2);
		}
}

// 2 is heating
if(outpercent2 <= 0)
{
		if(outpercent2 < -92.4)
		{
```

```
                    outpercent2 = -92.4;
                }

                digitalWrite(dir1, LOW); // current from B to A cooling
                digitalWrite(dir2, HIGH); // current from A to B heating

                x1 = 10824.94*(0.01*(outpercent1))-6.66;
                x2 = 10824.94*(0.01*(-outpercent2))-6.66;

                if(x1>x2)
                {
                        high_duration1 = x2;
                        high_duration2 = x1 - x2;

                        digitalWrite(pwm1, HIGH);
                        digitalWrite(pwm2, HIGH);
                        delayMicroseconds(high_duration1);

                        digitalWrite(pwm1, HIGH);
                        digitalWrite(pwm2, LOW);
                        delayMicroseconds(high_duration2);

                        digitalWrite(pwm1, LOW);
                        digitalWrite(pwm2, LOW);
                        delayMicroseconds(maxtime - x1);
                }

                else
                {
                        high_duration1 = x1;
                        high_duration2 = x2 - x1;

                        digitalWrite(pwm1, HIGH);
                        digitalWrite(pwm2, HIGH);
                        delayMicroseconds(high_duration1);

                        digitalWrite(pwm1, LOW);
                        digitalWrite(pwm2, HIGH);
                        delayMicroseconds(high_duration2);

                        digitalWrite(pwm1, LOW);
                        digitalWrite(pwm2, LOW);
                        delayMicroseconds(maxtime - x2);
                }
        }
}

// 1 is heating
if(outpercent1 <= 0)
{
```

```
if(outpercent1 < -92.4)
{
        outpercent1 = -92.4;
}

// 2 is cooling
if(outpercent2 > 0)
{
        if(outpercent2 > 92.4)
        {
                outpercent2 = 92.4;
        }

        digitalWrite(dir1, HIGH); // current from A to B heating
        digitalWrite(dir2, LOW); // current from B to A cooling

        x1 = 10824.94*(0.01*(-outpercent1))-6.66;
        x2 = 10824.94*(0.01*(outpercent2))-6.66;

        if(x1>x2)
        {
                high_duration1 = x2;
                high_duration2 = x1 - x2;

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration1);

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, LOW);
                delayMicroseconds(high_duration2);

                digitalWrite(pwm1, LOW);
                digitalWrite(pwm2, LOW);
                delayMicroseconds(maxtime - x1);
        }

        else
        {
                high_duration1 = x1;
                high_duration2 = x2 - x1;

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration1);

                digitalWrite(pwm1, LOW);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration2);
```

```
                digitalWrite(pwm1, LOW);
                digitalWrite(pwm2, LOW);
                delayMicroseconds(maxtime - x2);
        }
}

// 2 is heating
if(outpercent2 <= 0)
{
        if(outpercent2 < -92.4)
        {
                outpercent2 = -92.4;
        }

        digitalWrite(dir1, HIGH); // current from A to B heating
        digitalWrite(dir2, HIGH); // current from A to B heating

        x1 = 10824.94*(0.01*(-outpercent1))-6.66;
        x2 = 10824.94*(0.01*(-outpercent2))-6.66;

        if(x1>x2)
        {
                high_duration1 = x2;
                high_duration2 = x1 - x2;

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration1);

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, LOW);
                delayMicroseconds(high_duration2);

                digitalWrite(pwm1, LOW);
                digitalWrite(pwm2, LOW);
                delayMicroseconds(maxtime - x1);
        }

        else
        {
                high_duration1 = x1;
                high_duration2 = x2 - x1;

                digitalWrite(pwm1, HIGH);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration1);

                digitalWrite(pwm1, LOW);
                digitalWrite(pwm2, HIGH);
                delayMicroseconds(high_duration2);
```

```
                                        digitalWrite(pwm1, LOW);
                                        digitalWrite(pwm2, LOW);
                                        delayMicroseconds(maxtime - x2);
                                }
                        }
}

// Read the pressure
logp = analogRead(A3);
singlep = 0.01685*(double)logp;
sump = sump + singlep;

// Calculate temperature using thermistor
// http://www.circuitbasics.com/arduino-thermistor-temperature-sensor-tutorial/
D1=analogRead(A0); // Read voltage from voltage divider 1
D2=analogRead(A1); // Read voltage from voltage divider 2
D3=analogRead(A2); // Read voltage from voltage divider 2
R1=Rv1*(1023-D1)/D1; // Solve for resistance of thermistor 1 from voltage divider equation
R2=Rv2*(1023-D2)/D2; // Solve for resistance of thermistor 2 from voltage divider equation
R3=Rv3*(1023-D3)/D3; // Solve for resistance of thermistor 2 from voltage divider equation
lnR1=log(R1);
lnR2=log(R2);
lnR3=log(R3);
T1=319.72-36.471*lnR1+0.052763*lnR1*lnR1*lnR1; //Steinhart-Hart equation
T2=319.72-36.471*lnR2+0.052763*lnR2*lnR2*lnR2;
T3=319.72-36.471*lnR3+0.052763*lnR3*lnR3*lnR3;

// Calculute offset and adjust the output for top and bottom
T_offset1=T1-T_tar; // update temperature offset
T_offset2=T2-T_tar; // update temperature offset
outpercent1=T_offset1*P_value1; // update output percentage
outpercent2=T_offset2*P_value2; // update output percentage

// adjusting p-value when it's close to target temperature
if(T_offset1 >= 1)
{
        P_value1 = 1000;
}

if(T_offset1 <= -1)
{
        P_value1 = 1000;
}

if(T_offset1 <= 1)
{
        if(T_offset1 >= 0.1)
        {
                P_value1 = 100;
```

```
		}
}

if(T_offset1 >= -1)
{
		if(T_offset1 <= -0.1)
		{
			P_value1 = 100;
		}
}

if(T_offset1 <= 0.1)
{
		if(T_offset1 >= -0.1)
		{
			P_value1 = 10;
		}
}

if(T_offset2 >= 1)
{
P_value2 = 1000;
}

if(T_offset2 <= -1)
{
P_value2 = 1000;
}

if(T_offset2 <= 1)
{
		if(T_offset2 >= 0.1)
		{
			P_value2 = 100;
		}
}

if(T_offset2 >= -1)
{
		if(T_offset2 <= -0.1)
		{
			P_value2 = 100;
		}
}

if(T_offset2 <= 0.1)
{
		if(T_offset2 >= -0.1)
		{
			P_value2 = 10;
```

```
                }
        }

        if(outpercent1 > 92.4)
        {
                outpercent1 = 92.4;
        }

        if(outpercent1 < -92.4)
        {
                outpercent1 = -92.4;
        }

        if(outpercent2 > 92.4)
        {
                outpercent2 = 92.4;
        }

        if(outpercent2 < -92.4)
        {
                outpercent2 = -92.4;
        }
}

void DATALOGGING()
{
   count=count+1;
   if(count > log_period)
   {
                actualp = sump/((double)log_period ); // calculate an average pressure

                // Print to serial monitor
                Serial.print(rtc.getDateStr());
                Serial.print(" ");
                Serial.print(rtc.getTimeStr());
                Serial.print(" ");
                Serial.print(T1);
                Serial.print(" ");
                Serial.print(T2);
                Serial.print(" ");
                Serial.print(T3);
                Serial.print(" ");
                Serial.print(actualp);
                Serial.println(" ");

                // Print to LCD
                lcd.setCursor(5, 0);
                lcd.print(T1);
                lcd.setCursor(11, 0);
                lcd.print(T2);
```

```
                    lcd.setCursor(3, 1);
                    lcd.print(T3);
                    lcd.setCursor(11, 1);
                    lcd.print(actualp);

                    // Print to Micro-SD
                    myFile = SD.open("Data.txt", FILE_WRITE);
                    myFile.print(rtc.getDateStr());
                    myFile.print(" ");
                    myFile.print(rtc.getTimeStr());
                    myFile.print(" ");
                    myFile.print(T1);
                    myFile.print(" ");
                    myFile.print(T2);
                    myFile.print(" ");
                    myFile.print(T3);
                    myFile.print(" ");
                    myFile.print(actualp);
                    myFile.println("\n");
                    myFile.close();

                    count=1;
                    sump = 0;
   }
}

void loop()
{
        PIDControl();
        DATALOGGING();
}
```

Appendix B: MATLAB Code

B.1 Project 1: Water Saturation Map for WAG Analysis

```
% time in minutes corresponding to when ct images were taken during drainage (d) and imbibition (i) cycles
load time.mat;
load time1d.mat;
load time1i.mat;
load time2d.mat;
load time2i.mat;
load time3d.mat;
load time3i.mat;
load time4d.mat;

% total number of images for respective drainage and imbibition cycles
t1d=21;
t1i=26;
t2d=18;
t2i=26;
t3d=19;
t3i=26;
t4d=23;

% Variable zero matrix onto which individual CT images will be ascribed to in order to calculate water saturation. Dimensions of all our images are 1900 (W) x 1516 (H) pixel
drainage_1=zeros(1516,1900,t1d);
imbibition_1=zeros(1516,1900,t1i);
drainage_2=zeros(1516,1900,t2d);
imbibition_2=zeros(1516,1900,t2i);
drainage_3=zeros(1516,1900,t3d);
imbibition_3=zeros(1516,1900,t3i);
drainage_4=zeros(1516,1900,t4d);

% Variable zero matrix onto which our saturation calculations will be ascribed to. It is not the original 1900 x 1516 pixel size since the image is cropped to only include the rock slab
sw_drainage_1=zeros(841,1726,t1d);
sw_imbibition_1=zeros(841,1726,t1i);
sw_drainage_2=zeros(841,1726,t2d);
sw_imbibition_2=zeros(841,1726,t2i);
sw_drainage_3=zeros(841,1726,t3d);
sw_imbibition_3=zeros(841,1726,t3i);
sw_drainage_4=zeros(841,1726,t4d);

% dry and wet slab
dry=imread('jl_WAG_H4_dry_052218_9.tif');
wet=imread('jl_WAG_H4_wet_052218_10.tif');

% Performs median filtering, where each output pixel contains the median value in the 6-by-6 neighborhood around the corresponding pixel in the input image.
```

```
dry=medfilt2(dry,[6 6]);
wet=medfilt2(wet,[6 6]);

% To calculate water saturation, Sw = (RGx - RGdry) / (RGwet - RGdry)
denom=double(wet)-double(dry);

% Converts a numeric array into a character array that represents the numbers
strtime1d=num2str(time1d);
strtime1i=num2str(time1i);
strtime2d=num2str(time2d);
strtime2i=num2str(time2i);
strtime3d=num2str(time3d);
strtime3i=num2str(time3i);
strtime4d=num2str(time4d);

% Grayscale map for dry rock
figure;
set(gcf,'color','w');
hd=imagesc(double(dry(575:1415,50:1775,1)));
colormap('gray');
colorbar;
axis image;
saveas(hd,['Rock Bedding.tif'])

% The in situ change during 1st drainage
for i=1:t1d
str1d=['jl_WAG_H4_d1_052218_',num2str(i),'.tif'];
drainage_1(:,:,i)=imread(str1d);
drainage_1(:,:,i)=medfilt2(drainage_1(:,:,i),[6 6]);
sw_drainage_1(:,:,i)=(double(drainage_1(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
set(gcf,'color','w');
hd=imagesc(sw_drainage_1(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime1d(i,:), ' Minutes After 1st Drainage']);
saveas(hd,['drainage1_',num2str(i),'.tif']);
end

% The in situ change during 1st imbibition
for i=1:t1i
str1i=['jl_WAG_H4_i1_052218_',num2str(i),'.tif'];
imbibition_1(:,:,i)=imread(str1i);
imbibition_1(:,:,i)=medfilt2(imbibition_1(:,:,i),[6 6]);
sw_imbibition_1(:,:,i)=(double(imbibition_1(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
set(gcf,'color','w');
hd=imagesc(sw_imbibition_1(:,:,i),[0.2 1.4]);
colormap('Jet');
```

```
colorbar;
axis image;
title([strtime1i(i,:), ' Minutes After 1st Imbibition']);
saveas(hd,['imbibition1_',num2str(i),'.tif']);
end

% The in situ change during 2nd drainage
for i=1:t2d
str2d=['jl_WAG_H4_d2_052218_',num2str(i),'.tif'];
drainage_2(:,:,i)=imread(str2d);
drainage_2(:,:,i)=medfilt2(drainage_2(:,:,i),[6 6]);
sw_drainage_2(:,:,i)=(double(drainage_2(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
set(gcf,'color','w');
hd=imagesc(sw_drainage_2(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime2d(i,:), ' Minutes After 2nd Drainage']);
saveas(hd,['drainage2_',num2str(i),'.tif']);
end

% The in situ change during 2nd imbibition
for i=1:t2i
str2i=['jl_WAG_H4_i2_052218_',num2str(i),'.tif'];
imbibition_2(:,:,i)=imread(str2i);
imbibition_2(:,:,i)=medfilt2(imbibition_2(:,:,i),[6 6]);
sw_imbibition_2(:,:,i)=(double(imbibition_2(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
hd=imagesc(sw_imbibition_2(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime2i(i,:), ' Minutes After 2nd Imbibition']);
saveas(hd,['imbibition2_',num2str(i),'.tif']);
end
% The in situ change during 3rd drainage
for i=1:t3d
str3d=['jl_WAG_H4_d3_052218_',num2str(i),'.tif'];
drainage_3(:,:,i)=imread(str3d);
drainage_3(:,:,i)=medfilt2(drainage_3(:,:,i),[6 6]);
sw_drainage_3(:,:,i)=(double(drainage_3(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
set(gcf,'color','w');
hd=imagesc(sw_drainage_3(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime3d(i,:), ' Minutes After 3rd Drainage']);
saveas(hd,['drainage3_',num2str(i),'.tif']);
```

```
end

% The in situ change during 3rd imbibition
for i=1:t3i
str3i=['jl_WAG_H4_i3_052218_',num2str(i),'.tif'];
imbibition_3(:,:,i)=imread(str3i);
imbibition_3(:,:,i)=medfilt2(imbibition_3(:,:,i),[6 6]);
sw_imbibition_3(:,:,i)=(double(imbibition_3(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
hd=imagesc(sw_imbibition_3(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime3i(i,:), ' Minutes After 3rd Imbibition']);
saveas(hd,['imbibition3_',num2str(i),'.tif']);
end

% The in situ change during 4th drainage
for i=1:t4d
str4d=['jl_WAG_H4_d4_052218_',num2str(i),'.tif'];
drainage_4(:,:,i)=imread(str4d);
drainage_4(:,:,i)=medfilt2(drainage_4(:,:,i),[6 6]);
sw_drainage_4(:,:,i)=(double(drainage_4(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
hd=imagesc(sw_drainage_4(:,:,i),[0.2 1.4]);
colormap('Jet');
colorbar;
axis image;
title([strtime4d(i,:), ' Minutes After 4th Drainage']);
saveas(hd,['drainage4_',num2str(i),'.tif']);
end
toc
```

B.2 Project 1: Average Water Saturation for WAG Analysis

```
% The greyscale values of the dry radiography image is binarized by replacing all values above a globally
determined threshold with 1s and setting all other values to 2s. The tight sandstone rock corresponds to 2s
and the coarse sandstone rock corresponds to 1s.
crop = imcrop(dry,[100,575,1600,840]); % imcrop(X,[xmin ymin width height])
BW = imbinarize(crop,'adaptive','Sensitivity',0.605);
imshowpair(crop,BW,'montage')
grid2dBW_original = double(BW);
grid2dBW_original(grid2dBW_original==0)=2;
ncol=1601;
nrow=841;

% The in situ change during 1st drainage
for i=1:t1d
str1d=['jl_WAG_H4_d1_052218_',num2str(i),'.tif'];
drainage_1(:,:,i)=imread(str1d);
drainage_1(:,:,i)=medfilt2(drainage_1(:,:,i),[6 6]);
sw_drainage_1(:,:,i)=(double(drainage_1(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_drainage_1(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_d1(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_drainage_1=sw_drainage_1;
coarse_sw_drainage_1=sw_drainage_1;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_drainage_1(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_drainage_1(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_drainage_1(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_d1(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_drainage_1(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_d1(i,1)=mean(coarse_avg_sw_column,'omitnan');
end

% The in situ change during 1st imbibition
for i=1:t1i
str1i=['jl_WAG_H4_i1_052218_',num2str(i),'.tif'];
imbibition_1(:,:,i)=imread(str1i);
imbibition_1(:,:,i)=medfilt2(imbibition_1(:,:,i),[6 6]);
sw_imbibition_1(:,:,i)=(double(imbibition_1(575:1415,50:1775,i))-...
```

```
double(dry(575:1415,50:1775))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_imbibition_1(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_i1(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_imbibition_1=sw_imbibition_1;
coarse_sw_imbibition_1=sw_imbibition_1;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_imbibition_1(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_imbibition_1(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_imbibition_1(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_i1(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_imbibition_1(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_i1(i,1)=mean(coarse_avg_sw_column,'omitnan');
end

% The in situ change during 2nd drainage
for i=1:t2d
str2d=['jl_WAG_H4_d2_052218_',num2str(i),'.tif'];
drainage_2(:,:,i)=imread(str2d);
drainage_2(:,:,i)=medfilt2(drainage_2(:,:,i),[6 6]);
sw_drainage_2(:,:,i)=(double(drainage_2(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_drainage_2(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_d2(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_drainage_2=sw_drainage_2;
coarse_sw_drainage_2=sw_drainage_2;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_drainage_2(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_drainage_2(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_drainage_2(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
```

```
tight_avg_sw_d2(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_drainage_2(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_d2(i,1)=mean(coarse_avg_sw_column,'omitnan');
end

% The in situ change during 2nd imbibition
for i=1:t2i
str2i=['jl_WAG_H4_i2_052218_',num2str(i),'.tif'];
imbibition_2(:,:,i)=imread(str2i);
imbibition_2(:,:,i)=medfilt2(imbibition_2(:,:,i),[6 6]);
sw_imbibition_2(:,:,i)=(double(imbibition_2(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_imbibition_2(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_i2(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_imbibition_2=sw_imbibition_2;
coarse_sw_imbibition_2=sw_imbibition_2;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_imbibition_2(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_imbibition_2(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_imbibition_2(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_i2(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_imbibition_2(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_i2(i,1)=mean(coarse_avg_sw_column,'omitnan');
end

% The in situ change during 3rd drainage
for i=1:t3d
str3d=['jl_WAG_H4_d3_052218_',num2str(i),'.tif'];
drainage_3(:,:,i)=imread(str3d);
drainage_3(:,:,i)=medfilt2(drainage_3(:,:,i),[6 6]);
sw_drainage_3(:,:,i)=(double(drainage_3(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_drainage_3(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_d3(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_drainage_3=sw_drainage_3;
```

```
coarse_sw_drainage_3=sw_drainage_3;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_drainage_3(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_drainage_3(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_drainage_3(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_d3(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_drainage_3(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_d3(i,1)=mean(coarse_avg_sw_column,'omitnan');
end
% The in situ change during 3rd imbibition
for i=1:t3i
str2i=['jl_WAG_H4_i3_052218_',num2str(i),'.tif'];
imbibition_3(:,:,i)=imread(str2i);
imbibition_3(:,:,i)=medfilt2(imbibition_3(:,:,i),[6 6]);
sw_imbibition_3(:,:,i)=(double(imbibition_3(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_imbibition_3(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_i3(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_imbibition_3=sw_imbibition_3;
coarse_sw_imbibition_3=sw_imbibition_3;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_imbibition_3(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_imbibition_3(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_imbibition_3(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_i3(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_imbibition_3(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_i3(i,1)=mean(coarse_avg_sw_column,'omitnan');
end
```

```
% The in situ change during 4th drainage
for i=1:39
str3d=['jl_WAG_H4_d4_052218_',num2str(i),'.tif'];
drainage_4(:,:,i)=imread(str3d);
drainage_4(:,:,i)=medfilt2(drainage_4(:,:,i),[6 6]);
sw_drainage_4(:,:,i)=(double(drainage_4(575:1415,50:1775,i))-...
double(dry(575:1415,50:1775)))./denom(575:1415,50:1775);
% Calculating average water saturation for entire sample
avg_sw_column=mean(sw_drainage_4(:,:,i),'omitnan');
avg_sw_column(~isfinite(avg_sw_column))=NaN;
avg_sw_d4(i,1)=mean(avg_sw_column,'omitnan');
% Calculating average water saturation by rock type
tight_sw_drainage_4=sw_drainage_4;
coarse_sw_drainage_4=sw_drainage_4;
for c = 1:ncol
for r = 1:nrow
if grid2dBW_original(r,c) == 1;
tight_sw_drainage_4(r,c,i)=NaN;
else grid2dBW_original(r,c) == 2;
coarse_sw_drainage_4(r,c,i)=NaN;
end
end
end
tight_avg_sw_column=mean(tight_sw_drainage_4(:,:,i),'omitnan');
tight_avg_sw_column(~isfinite(tight_avg_sw_column))=NaN;
tight_avg_sw_d4(i,1)=mean(tight_avg_sw_column,'omitnan');
coarse_avg_sw_column=mean(coarse_sw_drainage_4(:,:,i),'omitnan');
coarse_avg_sw_column(~isfinite(coarse_avg_sw_column))=NaN;
coarse_avg_sw_d4(i,1)=mean(coarse_avg_sw_column,'omitnan');
end

% Plot average water saturation of entire rock sample and by bedding layer type as a function of time
figure;
avg_sw_all=vertcat(avg_sw_d1,avg_sw_i1,avg_sw_d2,avg_sw_i2,avg_sw_d3,avg_sw_i3,avg_sw_d4);
tight_avg_sw_all=vertcat(tight_avg_sw_d1,tight_avg_sw_i1,tight_avg_sw_d2,...
tight_avg_sw_i2,tight_avg_sw_d3,tight_avg_sw_i3,tight_avg_sw_d4);
coarse_avg_sw_all=vertcat(coarse_avg_sw_d1,coarse_avg_sw_i1,coarse_avg_sw_d2,...
coarse_avg_sw_i2,coarse_avg_sw_d3,coarse_avg_sw_i3,coarse_avg_sw_d4);
set(gcf,'color','white')
grid off;
hold on;
box on;
plot(time,avg_sw_all,'o',time,tight_avg_sw_all,':.',time,coarse_avg_sw_all,'--');
% Annotations
xlabel('Time, (seconds)');
ylabel('Water Saturation')
xlim([0 20]);
ylim([0 1]);
```

B.2 Project 3: Initial Gas Loading Pressure for Excess-Water Hydrate Experiment

```
% Calculate Initial Gas Loading Pressure (written by Tiannong 'Skyler' Dong)
P_Ex = fConvPres(1000,'psig','Pa'); % Experimental pressure in Pascal [SI]
T_Ex = fConvTemp(3,'C','K'); % Experimental temperature in Kelvin [SI]
T_Room = fConvTemp(22,'C','K'); % Room temperature during initial CH4 gas loading [SI]

R = 0.0079 * 0.5; % radius of porous medium [SI]
H = 0.0313; % height/thickness of porous medium [SI]
Porosity = 0.40; % porosity of porous medium
V_Pore = pi() * R^2 * H * Porosity; % pore volume [SI]
Density_GH = 0.9e+3; % density of gas hydrate [SI]

%% Molar mass [SI unit]
M_H2O = 18e+3; % molar mass of water [SI]
M_CH4 = 16e+3; % molar mass of methane [SI]
M_C2H6 = (12*2+6)*1e+3; % molar mass of ethane [SI]
% M_Gas = M_CH4;
M_Gas = M_CH4 * 0.9 + M_C2H6 * 0.1; % average molar mass of 90% methane and 10% ethane by
mole, assuming ideal gas mixing [SI]

%% Methane solubility [SI unit]
P_solu = P_Ex;
T_solu = T_Ex;
Salinity_solu = [44]'; % [input] unit: see next line
SalinityUnits_solu = 'ppt'; % parts per thousand
Datatype_solu = 'points';
NetCH4Solubility = fMethaneSolubility(P_solu,T_solu,Salinity_solu,SalinityUnits_solu,Datatype_solu);
%[mol/kg water]
% figure;
% plot(T,NetCH4Solubility,'bo')
% xlabel('Temperature (K)')
% ylabel('Methane Solubility (moles CH4 per kg water)')

mass_WaterSolution = 0.5e-3; %[SI, kg]
% mass_WaterSolution = V_Pore * 0.5; %[SI, kg]
mass_DissolvedCH4 = NetCH4Solubility .* mass_WaterSolution; % mass of dissolved CH4 [SI]

%%
S_GH = 0.4; % [input] assumed/targeted gas hydrate saturation after hydrate formation

V_GH = V_Pore * S_GH; % [SI]
mass_GH = V_GH * Density_GH; % mass of assumed/targeted gas hydrate
mass_Gas = mass_GH * M_Gas / (M_Gas + 5.75 * M_H2O) + mass_DissolvedCH4; % mass of gas in
assumed/targeted gas hydrate
Density_CH4 = fDensity_CH4(P_Ex,T_Ex); % [SI]

V_i = fConvVol(3,'mL','SI'); % [input] initial volume (chamber + dead space volume) [SI]
Density_CH4_i = mass_Gas ./ V_i; % desired density of gas at initial loading stage
```

```
Error = 1;
P_i = 1e6; % initial guess [Pa]
% P_step = P_i * 0.5;
counter = 0;

while Error >= 0.01
   Density_CH4_temp = fDensity_CH4(P_i, T_Room);
   Error = abs(Density_CH4_i - Density_CH4_temp) / Density_CH4_i;
   P_step = P_i * Error;
   P_i = P_i + P_step * ((Density_CH4_i - Density_CH4_temp)/abs(Density_CH4_i -
Density_CH4_temp));
   counter = counter + 1;
end

P_i_MPa = P_i * 1e-6;
P_i_psig = fConvPres(P_i,'SI','psig');

% disp('Pressure of initial CH4 is (MPa):');
Text = ['Initially load CH4 gas to: ', num2str(P_i_MPa),' MPa or ', num2str(P_i_psig),' psig.'];
disp(Text)

% ------------------------------------------------------------------------------------------------------------------

% Equation of State from Duan et al. (1992)
function Density_CH4 = fDensity_CH4(P,T)
R=83.1441;

c1= 4.30310345E+1;
c2=-6.83277221E-2;
c3=-5.68718730E+3;
c4= 3.56636821E-5;
c5=-5.79133791E+1;
c6= 6.11616662E-3;
c7=-7.85528103E-4;
c8=-9.42540759E-2;
c9= 1.92132040E-2;
c10= -9.17186899E-06;

lambda_c1=9.92230792E-2;
lambda_c2=2.57906811E-5;
lambda_c3=0.0;
lambda_c4=0.0;
lambda_c5=0.0;
lambda_c6=0.0;
lambda_c7=0.0;
lambda_c8=1.83451402E-2;
lambda_c9=0.0;
lambda_c10=-8.07196716E-6;

xi_c1=-6.23943799E-3;
```

```
xi_c2=0.0;
xi_c3=0.0;
xi_c4=0.0;
xi_c5=0.0;
xi_c6=0.0;
xi_c7=0.0;
xi_c8=0.0;
xi_c9=0.0;
xi_c10=0.0;

a1= 8.72553928E-2;
a2=-7.52599476E-1;
a3= 3.75419887E-1;
a4= 1.07291342E-2;
a5= 5.49626360E-3;
a6=-1.84772802E-2;
a7= 3.18993183E-4;
a8= 2.11079375E-4;
a9= 2.01682801E-5;
a10=-1.65606189E-5;
a11= 1.19614546E-4;
a12=-1.08087289E-4;
alpha= 4.48262295E-2;
beta= 7.53970000E-1;
gamma= 7.7167000E-2;

P=fConvPres(P,'Pa','bar'); % convert Pa to bar

% P is in bar and T is in K.

Tc=190.6;
Pc=46.41;
Pr=P/Pc;
Tr=T/Tc;

par=c1+c2*T+c3/T+c4*T^2+c5/(680-T)+...
   c6*P+c7*P*log(T)+c8*P/T+c9*P/(680-T)+c10*P^2/T;

lambda_par=lambda_c1+lambda_c2*T+lambda_c3/T+lambda_c4*T^2+lambda_c5/(680-T)+...
   lambda_c6*P+lambda_c7*P*log(T)+lambda_c8*P/T+lambda_c9*P/(680-T)+lambda_c10*P^2/T;

xi_par=xi_c1+xi_c2*T+xi_c3/T+xi_c4*T^2+xi_c5/(680-T)+...
   xi_c6*P+xi_c7*P*log(T)+xi_c8*P/T+xi_c9*P/(680-T)+xi_c10*P^2/T;

function Vr=calc_Vr(Pr,Tr)
x0=Tr/Pr;
x=x0;
loop=1;
iteration=0;
eps=1e-4;
```

```
while loop==1
   iteration=iteration+1;
   y0=duan(x,Pr,Tr);
   B=-y0;
   x=x+eps;
   y1=duan(x,Pr,Tr);
   A=(y1-y0)/eps;
   delta=A\B;
   x=x+delta;
   if iteration > 20
      loop=0;
   end
end
Vr=x;

Vr=calc_Vr(Pr,Tr);
Z=Pr*Vr/Tr;
tmp_v=Z*R*T/(P*1e5)/10;
Density_CH4=0.016/tmp_v;

% ------------------------------------------------------------------------------------------------------------------------

% Pressure Unit Conversion Function (written by Tiannong 'Skyler' Dong)
function DataOut = fConvPres(DataIn,FromUnit,ToUnit)

switch FromUnit
   case {'SI','Pa','pascal','Pascal','pascals','Pascals','PaA'}
      Offset1 = 0;
      ConvFactor1 = 1;
   case {'MPa','megapascal','Megapascal','megapascals','Megapascals','MPaA'}
      Offset1 = 0;
      ConvFactor1 = 1e+6;
   case {'MPag','MPaG'}
      Offset1 = 0.101325;
      ConvFactor1 = 1e+6;
   case {'bar','bars','Bar','Bars'}
      Offset1 = 1.01325;
      ConvFactor1 = 1e+5;
   case {'psig','PSIG','psiG'}
      Offset1 = 14.69595;
      ConvFactor1 = 6894.75729;
   case {'psi','psia','PSIA','psiA'}
      Offset1 = 0;
      ConvFactor1 = 6894.75729;
end

switch ToUnit
   case {'SI','Pa','pascal','Pascal','pascals','Pascals','PaA'}
      Offset2 = 0;
      ConvFactor2 = 1;
```

```
    case {'MPa','MPA','megapascal','Megapascal','megapascals','Megapascals','MPaA'}
        Offset2 = 0;
        ConvFactor2 = 1e-6;
    case {'MPag','MPAG','MPaG'}
        Offset2 = -0.101325;
        ConvFactor2 = 1e-6;
    case {'bar','bars','Bar','Bars'}
        Offset2 = -1.01325;
        ConvFactor2 = 1e-5;
    case {'psig','PSIG','psiG'}
        Offset2 = -14.69595;
        ConvFactor2 = 1/6894.75729;
    case {'psi','psia','PSIA','psiA'}
        Offset2 = 0;
        ConvFactor2 = 1/6894.75729;
end

DataSI = (DataIn + Offset1) .* ConvFactor1;
DataOut = (DataSI + Offset2) .* ConvFactor2;
end

% ---------------------------------------------------------------------------------------------------------------------

% Length Unit Conversion Function (written by Tiannong 'Skyler' Dong)
function DataOut = fConvLength(DataIn,FromUnit,ToUnit)

switch FromUnit
    case {'SI','m','M','meter','Meter','meters','Meters'}
        Offset1 = 0;
        ConvFactor1 = 1;
    case {'cm','CM','centimeter','Centimeter','centimeters','Centimeters'}
        Offset1 = 0;
        ConvFactor1 = 1e-2;
    case {'mm','MM','millimeter','millimeter','Millimeters','Millimeters'}
        Offset1 = 0;
        ConvFactor1 = 1e-3;
    case {'ft','FT','foot','feet','Foot','Feet'}
        Offset1 = 0;
        ConvFactor1 = 0.3048;
    case {'in','IN','inch','Inch','inches','Inches'}
        Offset1 = 0;
        ConvFactor1 = 0.0254;
end

switch ToUnit
    case {'SI','m','M','meter','Meter','meters','Meters'}
        Offset2 = 0;
        ConvFactor2 = 1;
    case {'cm','CM','centimeter','Centimeter','centimeters','Centimeters'}
        Offset2 = 0;
```

```
        ConvFactor2 = 1e+2;
    case {'mm','MM','millimeter','millimeter','Millimeters','Millimeters'}
        Offset2 = 0;
        ConvFactor2 = 1e+3;
    case {'ft','FT','foot','feet','Foot','Feet'}
        Offset2 = 0;
        ConvFactor2 = 3.28084;
    case {'in','IN','inch','Inch','inches','Inches'}
        Offset2 = 0;
        ConvFactor2 = 39.3701;
end

DataSI = (DataIn + Offset1) .* ConvFactor1;
DataOut = (DataSI + Offset2) .* ConvFactor2;

End

% ------------------------------------------------------------------------------------------------------------------------

% Temperature Unit Conversion Function (written by Tiannong 'Skyler' Dong)
function DataOut = fConvTemp(DataIn,FromUnit,ToUnit)

switch FromUnit
    case {'SI','K','Kelvin','kelvin'}
        Offset1 = 0;
        ConvFactor1 = 1;
    case {'C','°C','degC','Celsius','celsius','Centigrade','centigrade'}
        Offset1 = 273.15;
        ConvFactor1 = 1;
    case {'F','°F','degF','Fahrenheit','fahrenheit'}
        Offset1 = 459.67;
        ConvFactor1 = 5/9;
end

switch ToUnit
    case {'SI','K','Kelvin','kelvin'}
        Offset2 = 0;
        ConvFactor2 = 1;
    case {'C','°C','degC','Celsius','celsius','Centigrade','centigrade'}
        Offset2 = -273.15;
        ConvFactor2 = 1;
    case {'F','°F','degF','Fahrenheit','fahrenheit'}
        Offset2 = -459.67;
        ConvFactor2 = 9/5;
end

DataSI = (DataIn + Offset1) .* ConvFactor1;
DataOut = (DataSI + Offset2) .* ConvFactor2;

end
```

```
% ------------------------------------------------------------------------------------------------------------------

% Volume Unit Conversion Function (written by Tiannong 'Skyler' Dong)
function DataOut = fConvVol(DataIn,FromUnit,ToUnit)

switch FromUnit
   case {'SI','m3','m^3','meter3','meters3','meter^3','meters^3','cubic meter','Cubic Meter','cubic
meters','Cubic Meters'}
      Offset1 = 0;
      ConvFactor1 = 1;
   case {'cm3','CM3','cm^3','CM^3','cubic centimeter','cubic centimeters','Cubic Centimeter','Cubic
Centimeters','mL','ML','millimeter','Millimeter','millimeters','Millimeters'}
      Offset1 = 0;
      ConvFactor1 = 1e-6;
end

switch ToUnit
   case {'SI','m3','m^3','meter3','meters3','meter^3','meters^3','cubic meter','Cubic Meter','cubic
meters','Cubic Meters'}
      Offset2 = 0;
      ConvFactor2 = 1;
   case {'cm3','cm^3','cubic centimeter','cubic centimeters','Cubic Centimeter','Cubic
Centimeters','mL','millimeter','millimeters','millimeters','Millimeters'}
      Offset2 = 0;
      ConvFactor2 = 1e+6;
end

DataSI = (DataIn + Offset1) .* ConvFactor1;
DataOut = (DataSI + Offset2) .* ConvFactor2;

end
```

```
% ------------------------------------------------------------------------------------------------------------------
% Function for Calculating the Solubility of Methane in Water as a Function of Pressure, Temperature
and Salinity (originally written by William Waite)

function NetCH4Solubility = fMethaneSolubility(P,T,Salinity,SalinityUnits,Datatype)
% In the presence of hydrate, the solubility is calculated according to Tishchenko et al. (2005)
% At pressures and temperatures for which hydrate does not form, the methane solubilities are calculated
based on two works by Duan et al. (1992 and 2006)

P = P*1e-6; %convert Pascal to MPa

[Smolality, Sppt] = SalinityConverter(Salinity,SalinityUnits);
% Provides the program with molality and ppt units for salinity

   if strcmp(Datatype,'range')
      progressbar('Salinity', 'Pressure')
         CH4Solubility_hydrate = zeros(length(P),length(T),length(Salinity));
         CH4Solubility_nohydrate = zeros(length(P),length(T),length(Salinity));
      for j = 1:length(Salinity)
         CH4Solubility_hydrate(:,:,j) = Tishchenko_2005(P,T,Sppt(j),Datatype);
         % From Tishchenko et al. (2005)
         for k = 1: length(P)
            CH4Solubility_nohydrate(k,:,j) = Duan_1992_2006(P(k)*10,T,Smolality(j));
            progressbar([],k/length(P))
         end
         progressbar(j/length(Salinity), [])
      end
   else
      progressbar('Datapoints')
         CH4Solubility_hydrate = zeros(length(P));
         CH4Solubility_nohydrate_temp = zeros(length(P));
      CH4Solubility_hydrate = Tishchenko_2005(P,T,Sppt,Datatype);
      for k = 1: length(P)
          CH4Solubility_nohydrate_temp(k) = Duan_1992_2006(P(k)*10,T(k),Smolality(k));
         progressbar(k/length(P))
      end
      CH4Solubility_nohydrate = CH4Solubility_nohydrate_temp(:,1);
   end

NetCH4Solubility = min(CH4Solubility_hydrate, CH4Solubility_nohydrate);

function [Smolality, Sppt] = SalinityConverter(S,units)
% Converts between molality (Smolality, in moles of salt per kilogram water) and salinity (Sppt, in grams
of salt per kg of solution (water + salt)) assuming only NaCl is present in the water.
   Msalt = 58.4428;          % Grams of salt (NaCl) per mole
   if strcmp(units,'mol/kg')          %if units == 'mol/kg'
      Smolality = S;
      Sppt = (S.*Msalt)./(1 + (S.*Msalt/1000));   % Molality to Salinity
   else
      Smolality = S./(Msalt.*(1-S./1000));           % Salinity to Molality
```

```
        Sppt = S;
    end
end

function [CH4SolAtPress] = Tishchenko_2005(P,T,S,Datatype)

    % Calculate the methane hydrate dissociation pressure, Pdiss (MPa)
    lnPdiss = -1.6444866e3 - 0.1374178.*T + (5.4979866e4)./T + (2.64118188e2).*log(T) + ...
        S.*(1.1178266e4 + 7.67420344.*T - (4.515213e-3).*(T.^2) - (2.04872879e5)./T - ...
        (2.17246046e3).*(log(T))) + (S.^2).*(1.70484431e2 + 0.118594073.*T - ...
        (7.0581304e-5).*(T.^2) - (3.09796169e3)./T - 33.2031996.*log(T));
    Pdiss = exp(lnPdiss);

    % Calculate the methane solubility in the presence of methane hydrate, Chyd (mol/kg)
    lnChyd = -2.5640213e5 - (1.6448053e2).*T + (9.1089042e-2).*(T.^2) + (4.90352929e6)./T + ...
        (4.93009113e4).*log(T) + S.*(-5.16285134e2 - 0.33622376.*T + (1.88199047e-4).*(T.^2) + ...
        (9.76525718e3)./T + 9.9523354e1.*log(T))

    % Calculate the methane solubility in the presence of methane hydrate at arbitrary pressure
    if strcmp(Datatype,'range')
            CH4SolAtPress = zeros(length(P),length(T));
        for i = 1: length(P)
        lnChydpress = lnChyd + ((5.04597e-2) + (7.64415e-4).*S - ((3.90236e-4) + (5.48947e-6).*S).*T + ...
            ((7.06154e-7) + (9.87742e-9).*S).*(T.^2)).*(P(i) - Pdiss) + ((7.57285e-5) - (1.90867e-8).*S - ...
            (1.4483e-10).*(S.^2) - ((1.96207e-7) - (6.67456e-11).*S).*T).*((P(i)-Pdiss).^2);
        CH4SolAtPress(i,:) = exp(lnChydpress);
        end
    else
        lnChydpress = lnChyd + ((5.04597e-2) + (7.64415e-4).*S - ((3.90236e-4) + (5.48947e-6).*S).*T + ...
            ((7.06154e-7) + (9.87742e-9).*S).*(T.^2)).*(P - Pdiss) + ((7.57285e-5) - (1.90867e-8).*S - ...
            (1.4483e-10).*(S.^2) - ((1.96207e-7) - (6.67456e-11).*S).*T).*((P - Pdiss).^2);
        CH4SolAtPress = exp(lnChydpress);
    end

end

function CH4solubility_nohydrate = Duan_1992_2006(P,T,mNa)
    % Calculate xCH4, the mole fraction of methane in the gas phase (Equation 4 in Duan_1992).
    PartialpressureH2O = zeros(length(T),1);
    for i = 1:length(T)
        if T(i) > 273.15
            PartialpressureH2O(i) = XSteam_p4_T(T(i));
        else
            PartialpressureH2O(i) = .0061;
        end
    end
    xCH4 = (P - PartialpressureH2O)./P;

    % Calculate the fugacity coefficient phiCH4
    phiCH4 = fugacitycoefficientCH4(P,T);
```

```
    % Calculate the interaction parameters needed for the solubility equation
    [muCH4RT,lambdaCH4Na,chiCH4NaCl] = interactionparameters_2006(P,T);
        mK = 0;
        mCa = 0;
        mMg = 0;
        mSO4 = 0;
        mCl = mNa;
        lambdaCH4SO4 = 0.0332;
    lnmCH4 = log(xCH4.*phiCH4.*P) - muCH4RT - 2*lambdaCH4Na.*(mNa + mK + 2*mCa + 2*mMg)-
chiCH4NaCl.*(mNa + mK + 2*mCa + 2*mMg).*(mCl + 2.*mSO4) - 4.*lambdaCH4SO4*mSO4;
    CH4solubility_nohydrate = exp(lnmCH4);
end

function PhiCH4=fugacitycoefficientCH4(P,T)

    % Fit parameters from Table A1 of Duan_1992
    a1 = 8.72553928e-2;
    a2 = -7.52599476e-1;
    a3 = 3.75419887e-1;
    a4 = 1.07291342e-2;
    a5 = 5.49626360e-3;
    a6 = -1.84772802e-2;
    a7 = 3.18993183e-4;
    a8 = 2.11079375e-4;
    a9 = 2.01682801e-5;
    a10 = -1.65606189e-5;
    a11 = 1.19614546e-4;
    a12 = -1.08087289e-4;
    alpha = 4.48262295e-2;
    beta = 7.5397e-1;
    gamma = 7.7167e-2;

    % Critical Temperature (Tc) and Pressure (Pc) from Appendix of Duan_1992
    Tc = 190.6;        % In K
    Pc = 46.41;        % In bar

    % Convenient parameter clusters for the equation of state (Used in both Duan 1992a and b)

    Pr = (P)/Pc;        % Pressure needs to be in bar for this calculation
    Tr = (T)/Tc;        % T needs to be in Kelvin for the equation of state calculation
    VrSeed = 1;         % This is a total guess.

    B = a1 + a2./(Tr.^2) + a3./(Tr.^3);
    C = a4 + a5./(Tr.^2) + a6./(Tr.^3);
    D = a7 + a8./(Tr.^2) + a9./(Tr.^3);
    E = a10 + a11./(Tr.^2) + a12./(Tr.^3);
    F = alpha./(Tr.^3);
```

```
    % Obtain the appropriate Vr and compressibility factor, Z by equating the left and right-hand sides of equation A1 in Duan_1992
    Vr = zeros(length(T),1);
    for i = 1: length(T)
        Vrtemp = VrSeed;
        Vrtemp = fminsearch(@Vrfinder,Vrtemp);
        Vr(i) = Vrtemp;
    end
    Z = 1 + B./Vr + C./(Vr.^2) + D./(Vr.^4) + E./(Vr.^5) + (F./(Vr.^2)).*(beta + gamma./(Vr.^2)).*exp(-gamma./(Vr.^2));

    % Obtain the methane fugacity, PhiCH4 from equation A2 in Duan_1992
    lnPhiCH4 = Z - 1 - log(Z) + B./Vr + C./(2*(Vr.^2)) + D./(4*(Vr.^4)) + E./(5*(Vr.^5)) + (F./(2*gamma)).*(beta + 1 - (beta + 1 + gamma./(Vr.^2)).*exp(-gamma./(Vr.^2)));
    PhiCH4 = exp(lnPhiCH4);

    % Numerically calculate Vr
    function minerrorVr = Vrfinder(Vrtemp)
        Z = 1 + B(i)./Vrtemp + C(i)./(Vrtemp.^2) + D(i)./(Vrtemp.^4) + E(i)./(Vrtemp.^5) + (F(i)./(Vrtemp.^2)).*(beta + gamma./(Vrtemp.^2)).*exp(-gamma./(Vrtemp.^2));
        minerrorVr = (((Pr.*Vrtemp./Tr(i)) - Z)*1000).^2;
    end
end

function [muCH4RT,lambdaCH4Na,chiCH4NaCl] = interactionparameters_2006(P,T)
    % This function uses the fit parameters in Table 3 of Duan_2006 to calculate interaction parameters using Equation 9 in Duan_2006
    c1 = [0.83143711e1, -0.81222036, -0.29903571e-02];
    c2 = [-0.72772168e-3, 0.10635172e-2, 0];
    c3 = [0.21489858e4, 0.18894036e3, 0];
    c4 = [-0.14019672e-4, 0, 0];
    c5 = [-0.66743449e6, 0, 0];
    c6 = [0.76985890e-2, 0.44105635e-4, 0];
    c7 = [-0.50253331e-5, 0, 0];
    c8 = [-0.30092013e1, 0, 0];
    c9 = [0.48468502e3, 0, 0];
    c10 = [0, -0.46797718e-10, 0];

    muCH4RT = c1(1) + c2(1)*T + c3(1)./T + c4(1)*(T.^2) + c5(1)./(T.^2) + c6(1)*P + c7(1)*P.*T + c8(1)*P./T + c9(1)*P./(T.^2) + c10(1).*(P.^2).*T;
    lambdaCH4Na = c1(2) + c2(2)*T + c3(2)./T + c4(2)*(T.^2) + c5(2)./(T.^2) + c6(2)*P + c7(2)*P.*T + c8(2)*P./T + c9(2)*P./(T.^2) + c10(2).*(P.^2).*T;
    chiCH4NaCl = c1(3);
end

function p4_T2 = XSteam_p4_T(T)
    teta = T - 0.23855557567849 / (T - 650.17534844798);
    a = teta ^ 2 + 1167.0521452767 * teta - 724213.16703206;
    B = -17.073846940092 * teta ^ 2 + 12020.82470247 * teta - 3232555.0322333;
    C = 14.91510861353 * teta ^ 2 - 4823.2657361591 * teta + 405113.40542057;
```

```
    p4_T2 = ((2 * C / (-B + (B ^ 2 - 4 * a * C) ^ 0.5)) ^ 4)*10;
end
function progressbar(varargin)
persistent progfig progdata lastupdate

% Get inputs
if nargin > 0
    input = varargin;
    ninput = nargin;
else
    % If no inputs, init with a single bar
    input = {0};
    ninput = 1;
end

% If task completed, close figure and clear vars, then exit
if input{1} == 1
    if ishandle(progfig)
        delete(progfig) % Close progress bar
    end
    clear progfig progdata lastupdate % Clear persistent vars
    drawnow
    return
end

% Init reset flag
resetflag = false;

% Set reset flag if first input is a string
if ischar(input{1})
    resetflag = true;
end

% Set reset flag if all inputs are zero
if input{1} == 0
    % If the quick check above passes, need to check all inputs
    if all([input{:}] == 0) && (length([input{:}]) == ninput)
        resetflag = true;
    end
end

% Set reset flag if more inputs than bars
if ninput > length(progdata)
    resetflag = true;
end

% If reset needed, close figure and forget old data
if resetflag
    if ishandle(progfig)
        delete(progfig) % Close progress bar
```

```
    end
    progfig = [];
    progdata = []; % Forget obsolete data
end

% Create new progress bar if needed
if ishandle(progfig)
else % This strange if-else works when progfig is empty (~ishandle() does not)

    % Define figure size and axes padding for the single bar case
    height = 0.03;
    width = height * 8;
    hpad = 0.02;
    vpad = 0.25;

    % Figure out how many bars to draw
    nbars = max(ninput, length(progdata));

    % Adjust figure size and axes padding for number of bars
    heightfactor = (1 - vpad) * nbars + vpad;
    height = height * heightfactor;
    vpad = vpad / heightfactor;

    % Initialize progress bar figure
    left = (1 - width) / 2;
    bottom = (1 - height) / 2;
    progfig = figure(...
        'Units', 'normalized',...
        'Position', [left bottom width height],...
        'NumberTitle', 'off',...
        'Resize', 'off',...
        'MenuBar', 'none' );

    % Initialize axes, patch, and text for each bar
    left = hpad;
    width = 1 - 2*hpad;
    vpadtotal = vpad * (nbars + 1);
    height = (1 - vpadtotal) / nbars;
    for ndx = 1:nbars
        % Create axes, patch, and text
        bottom = vpad + (vpad + height) * (nbars - ndx);
        progdata(ndx).progaxes = axes( ...
            'Position', [left bottom width height], ...
            'XLim', [0 1], ...
            'YLim', [0 1], ...
            'Box', 'on', ...
            'ytick', [], ...
            'xtick', [] );
        progdata(ndx).progpatch = patch( ...
            'XData', [0 0 0 0], ...
```

```
            'YData', [0 0 1 1] );
        progdata(ndx).progtext = text(0.99, 0.5, '', ...
            'HorizontalAlignment', 'Right', ...
            'FontUnits', 'Normalized', ...
            'FontSize', 0.7 );
        progdata(ndx).proglabel = text(0.01, 0.5, '', ...
            'HorizontalAlignment', 'Left', ...
            'FontUnits', 'Normalized', ...
            'FontSize', 0.7 );
        if ischar(input{ndx})
            set(progdata(ndx).proglabel, 'String', input{ndx})
            input{ndx} = 0;
        end

        % Set callbacks to change color on mouse click
        set(progdata(ndx).progaxes, 'ButtonDownFcn', {@changecolor, progdata(ndx).progpatch})
        set(progdata(ndx).progpatch, 'ButtonDownFcn', {@changecolor, progdata(ndx).progpatch})
        set(progdata(ndx).progtext, 'ButtonDownFcn', {@changecolor, progdata(ndx).progpatch})
        set(progdata(ndx).proglabel, 'ButtonDownFcn', {@changecolor, progdata(ndx).progpatch})

        % Pick a random color for this patch
        changecolor([], [], progdata(ndx).progpatch)

        % Set starting time reference
        if ~isfield(progdata(ndx), 'starttime') || isempty(progdata(ndx).starttime)
            progdata(ndx).starttime = clock;
        end
    end

    % Set time of last update to ensure a redraw
    lastupdate = clock - 1;

end

% Process inputs and update state of progdata
for ndx = 1:ninput
    if ~isempty(input{ndx})
        progdata(ndx).fractiondone = input{ndx};
        progdata(ndx).clock = clock;
    end
end

% Enforce a minimum time interval between graphics updates
myclock = clock;
if abs(myclock(6) - lastupdate(6)) < 0.01 % Could use etime() but this is faster
    return
end

% Update progress patch
for ndx = 1:length(progdata)
```

```
    set(progdata(ndx).progpatch, 'XData', ...
        [0, progdata(ndx).fractiondone, progdata(ndx).fractiondone, 0])
end

% Update progress text if there is more than one bar
if length(progdata) > 1
    for ndx = 1:length(progdata)
        set(progdata(ndx).progtext, 'String', ...
            sprintf('%1d%%', floor(100*progdata(ndx).fractiondone)))
    end
end

% Update progress figure title bar
if progdata(1).fractiondone > 0
    runtime = etime(progdata(1).clock, progdata(1).starttime);
    timeleft = runtime / progdata(1).fractiondone - runtime;
    timeleftstr = sec2timestr(timeleft);
    titlebarstr = sprintf('%2d%%    %s remaining', ...
        floor(100*progdata(1).fractiondone), timeleftstr);
else
    titlebarstr = ' 0%';
end
set(progfig, 'Name', titlebarstr)

% Force redraw to show changes
drawnow

% Record time of this update
lastupdate = clock;
end

% ------------------------------------------------------------------------
function changecolor(h, e, progpatch) %#ok<INUSL>
% Change the color of the progress bar patch

% Prevent color from being too dark or too light
colormin = 1.5;
colormax = 2.8;

thiscolor = rand(1, 3);
while (sum(thiscolor) < colormin) || (sum(thiscolor) > colormax)
    thiscolor = rand(1, 3);
end

set(progpatch, 'FaceColor', thiscolor)
end

% ------------------------------------------------------------------------
function timestr = sec2timestr(sec)
% Convert a time measurement from seconds into a human readable string.
```

```
% Convert seconds to other units
w = floor(sec/604800); % Weeks
sec = sec - w*604800;
d = floor(sec/86400); % Days
sec = sec - d*86400;
h = floor(sec/3600); % Hours
sec = sec - h*3600;
m = floor(sec/60); % Minutes
sec = sec - m*60;
s = floor(sec); % Seconds

% Create time string
if w > 0
   if w > 9
      timestr = sprintf('%d week', w);
   else
      timestr = sprintf('%d week, %d day', w, d);
   end
elseif d > 0
   if d > 9
      timestr = sprintf('%d day', d);
   else
      timestr = sprintf('%d day, %d hr', d, h);
   end
elseif h > 0
   if h > 9
      timestr = sprintf('%d hr', h);
   else
      timestr = sprintf('%d hr, %d min', h, m);
   end
elseif m > 0
   if m > 9
      timestr = sprintf('%d min', m);
   else
      timestr = sprintf('%d min, %d sec', m, s);
   end
else
   timestr = sprintf('%d sec', s);
end
end
end
```

Appendix C: Syringe Pump

Commercial syringe pumps, while available, were not suitable for the applications of the first project. The X-ray machine is a shielded enclosure, providing a physical barrier between radioactive X-rays and the outside environment. This means the X-ray cabinet is a self-contained and closed system. Since it is not possible to run a power outlet from within the shielded enclosure, any pumps or tools placed within the X-ray cabinet can only be powered from within the X-ray cabinet via batteries. Once the interlocks of the X-ray cabinet are secured, which is the only way for X-rays to be generated, users will also not have access to the interior of the X-ray cabinet. Thus, the pump along with the rest of the injection apparatus must be remotely controlled via Bluetooth.

A syringe pump was modified and designed from open source instructions provided by Naroom (2014). The Arduino IDE sketch for controlling the stepper motor via Bluetooth was created with reference to tutorials created by Nedelkovski (2017). The syringe pump is able to inject at pressures as high as 100 psi and at flow rates as low as 1 cc/min. Total injection volume before having to disassembly and refill the pump is approximately 50 mL. The hardware and electrical components for the syringe pump are listed in section **C.1** and **C.2**, respectively.

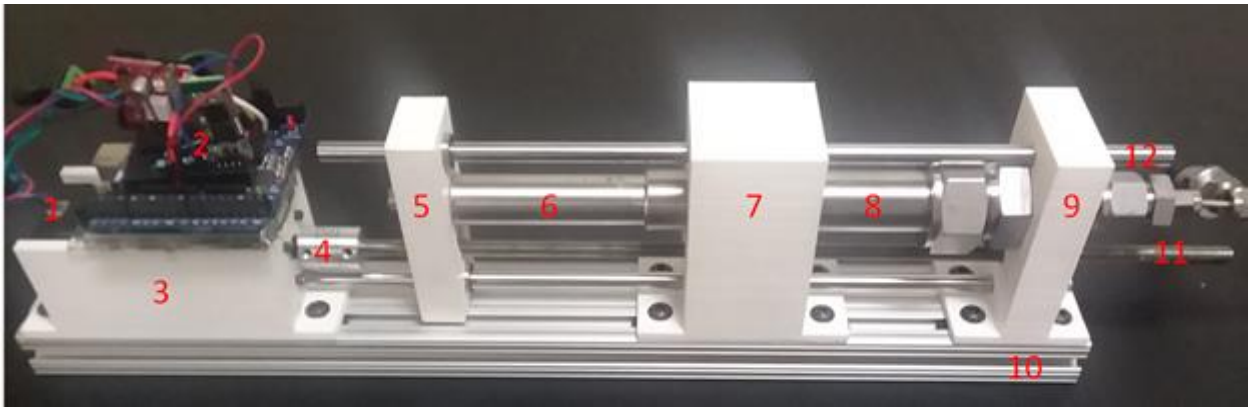

Fig. C.1—Syringe pump with hardware components (1) NEMA motor (2) electronics (3) motor mount (4) shaft coupler (5) syringe plunger mount (6) syringe plunger (7) syringe barrel holder (8) syringe barrel (9) syringe tip holder (10) mounting rail (11) M8 threaded rod (12) smooth rod.

C.1 Hardware for Syringe Pump

1) NEMA Stepper Motor

A NEMA 17 stepper motor with a gear ratio of 100:1 is used to achieve the required torque and speed (**Fig. C.2**). With a 42mm body and 1.68A rated current, the motor purchased from STEPPERONLINE is very suitable for an application that requires low speed and high torque up to 400 N·cm. Stepper motor is a 2-phase motor with a stepping angle of 0.018 degrees, efficiency of 73%, and operating voltage of 2.8V.

Fig. C.2—Syringe pump component: NEMA motor.

With a gear ratio of 100:1, the motor has 20000 steps per revolution or 0.018 degrees for each step. Without the gear box, the motor would only have 200 steps per revolution. The micro-stepping driver allows 16 micro-steps in one stepper motor step. Since the M8 threaded rod to which the motor is attached has a 1.25 mm pitch, this means for every 360 degrees rotation of the threaded rod, the M8 nut screw will dislocate 1.25mm. Recall that the nut screw is embedded inside the syringe plunger mount. This part translates the rotational movement of the stepper motor and the attached M8 threaded rod to the horizontal movement used to push or pull the syringe.

The calculation for the number of micro-steps for 1 mm of horizontal movement is shown with **Eq. C-1**. The calculation for the number of micro-steps for 1 mL of syringe volume displacement is shown with **Eq. C-2**.

$$MicroSteps1MM = \frac{MicroSteps\_Per\_Step * Steps\_Per\_Revolution}{Threaded\_Rod\_Pitch} \quad \text{(C-1)}$$

$$MicroSteps1ML = MicroSteps1MM * \frac{Syringe\_Barrel\_Length}{Syringe\_Volume\_Length} \quad \text{(C-2)}$$

With a syringe volume of 53 mL, barrel diameter of 21.20 mm, and barrel length of 150 mm, for every 360 degrees rotation of the threaded rod or 256000 micro-steps, the M8 nut screw will displace by 1 mm. For 724527 micro-steps, the syringe volume will displace by 1 mL.

The stepping signal duration directly determines the rate at which volume is displaced. For example if the stepping signal lasts 1 ms high and 1 ms low (each complete pulse taking 2 ms of time), then this equates to 500 microsteps per second. If the stepping signal lasts 10 ms high and 10 ms low (each complete pulse taking 20 ms of time), then this equates to 50 microsteps per second. Hence, the delay required to achieve the desired stepping signal duration, which in turn determines the desired rate of injection, is calculated using **Eq. C-3**.

$$MicroSecondDelay = \frac{Gearbox\ Inefficency * 10^6}{(2 * (Desired\_Injection\_Rate * MicroSteps1ML)/60)} \quad \text{(C-3)}$$

With a gearbox inefficiency of 27% and desired injection rate of 1 cc/min, then the required delay is calculated to be 11.18 ms. This delay is set within the Arduino IDE environment if the user specifies an injection rate of 1 cc/min, either via Android app or through the computer using Tera Term. More details follow in Section C.3 and C.4.

2) Electronics

The various electrical components include a separate Arduino Uno, HC-06 Bluetooth module, Big Easy Driver, and Proto-Screwshield (Wingshield) for Arduino Uno. Together, these electronics allow control of syringe pump via Bluetooth connectivity. Detailed explanation of the electronics is described in Section **C.2**.

3) Motor Mount

The 3D printed part used to securely hold the NEMA motor in place. 3D printed parts were printed in ABS (Acrylonitrile Butadiene Styrene) thermoplastic and designed using TinkerCad and OpenScad.

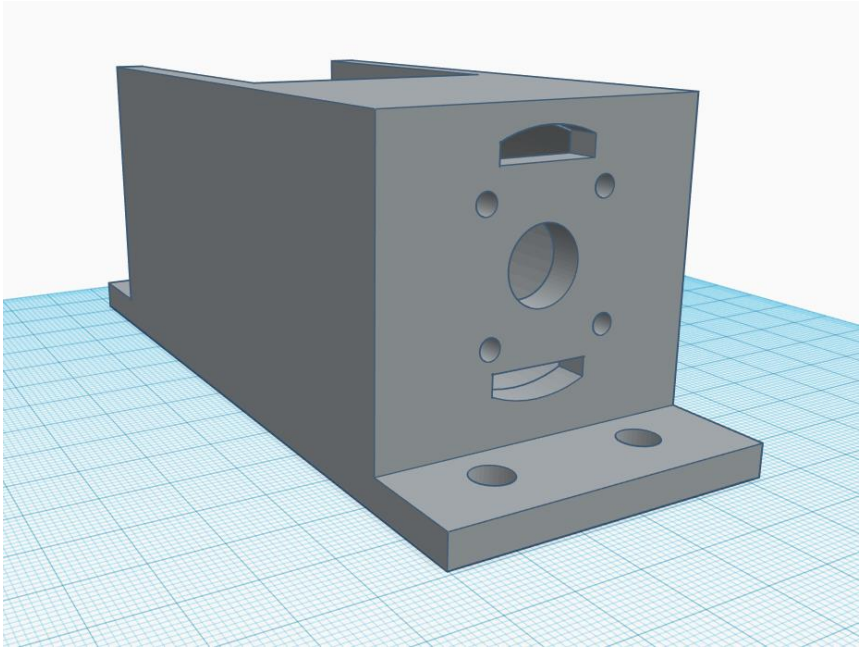

Fig. C.3—Syringe pump component: motor mount.

4) Shaft Coupler

Purchased from STEPPERONLINE, the 8mm-8mm Flexible Coupling 18x25mm CNC Stepper Motor Shaft Coupler connects the stepper motor to the threaded rod.

Fig. C.4—Syringe pump component: shaft coupler.

5) Syringe Plunger Mount

The 3D printed part with a M8 nut embedded inside used to translate the rotational movement of the stepper motor and the attached M8 threaded rod to the translational, horizontal movement used to push or pull the syringe.

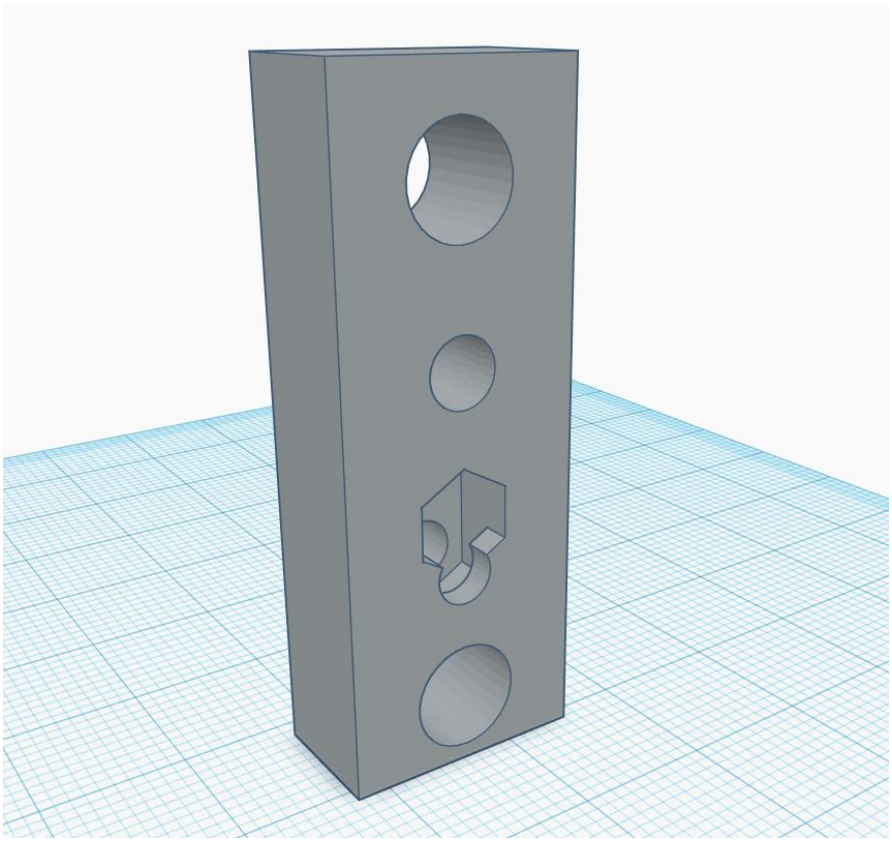

Fig. C.5—Syringe pump component: syringe plunger mount.

6) Syringe Plunger

The stainless steel part of the syringe that pushes against the liquid residing in the syringe barrel, which is hollow and filled with liquid.

7) Syringe Barrel Holder

The 3D printed part used to hold the syringe barrel in place.

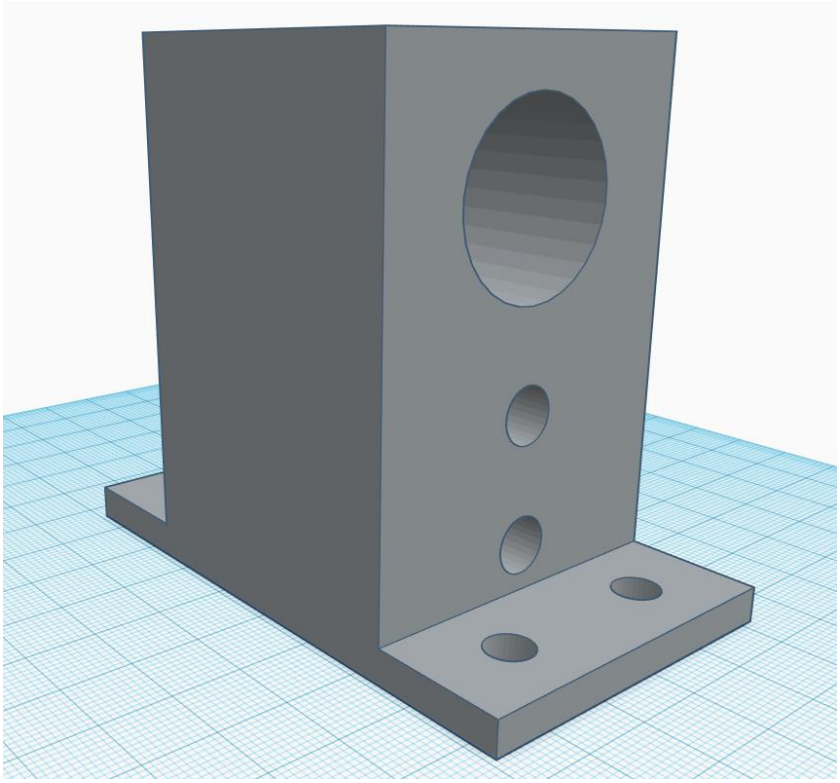

Fig. C.6—Syringe pump component: syringe barrel holder.

8) Syringe Barrel

The stainless steel part of the syringe that holds the injection liquid. Syringe volume is approximately 53 mL, barrel diameter of 21.20 mm, and barrel length of 150 mm.

9) Syringe Tip Holder

The 3D printed part that rests against the syringe tip. Only the syringe tip holder has to be unscrewed and removed in order to manually refill the syringe barrel.

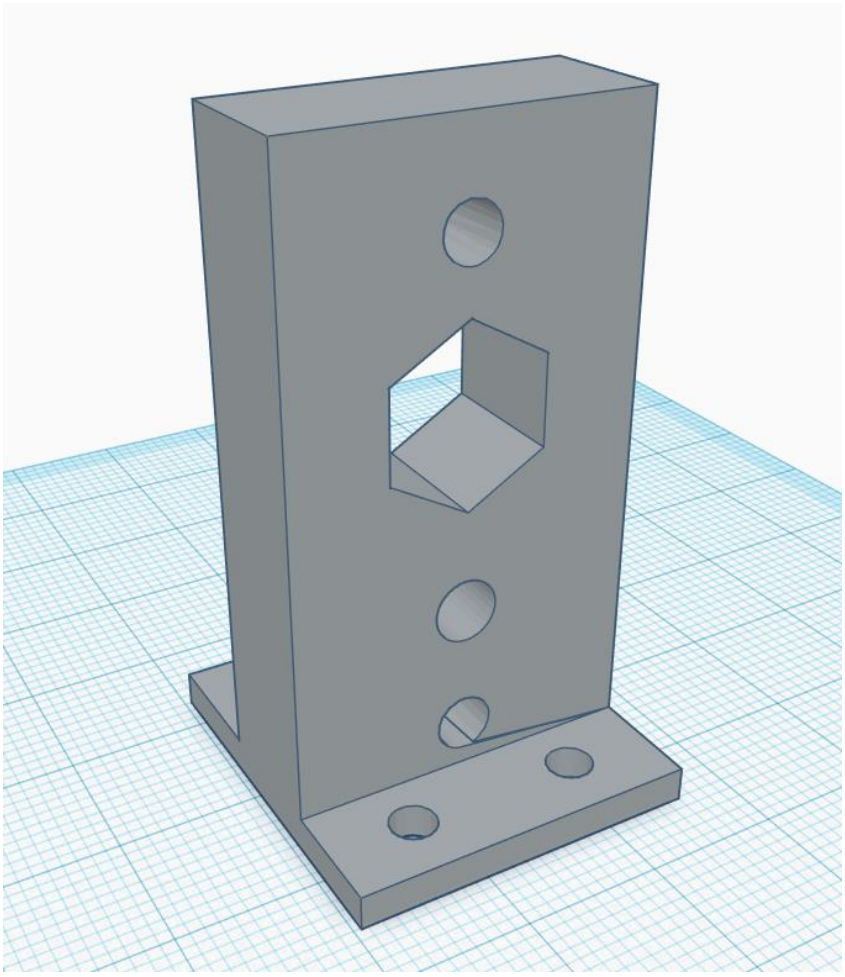

Fig. C.7—Syringe pump component: syringe tip holder.

10) Mounting Rail

Components are mounted onto an 18-inch piece of 1" by 2" 80/20 extruded aluminum rail, which fastens the modular design in place.

11) M8 Threaded Rod

The rod that rotates with the stepper motor. It is used to translate the rotational movement of the stepper motor to the translational, horizontal movement used to push or pull the syringe.

12) Smooth Rod

Two smooth rods hold the separate components in place.

C.2 Electronics for Syringe Pump

The various electrical components together allow a user to use an Android phone or computer to remotely control the syringe pump via Bluetooth connectivity. The user can specify the desired injection rate and motor direction. Details covered in Section C.3 and C.4. A breadboard diagram of the electronics is shown in **Fig. C.8**.

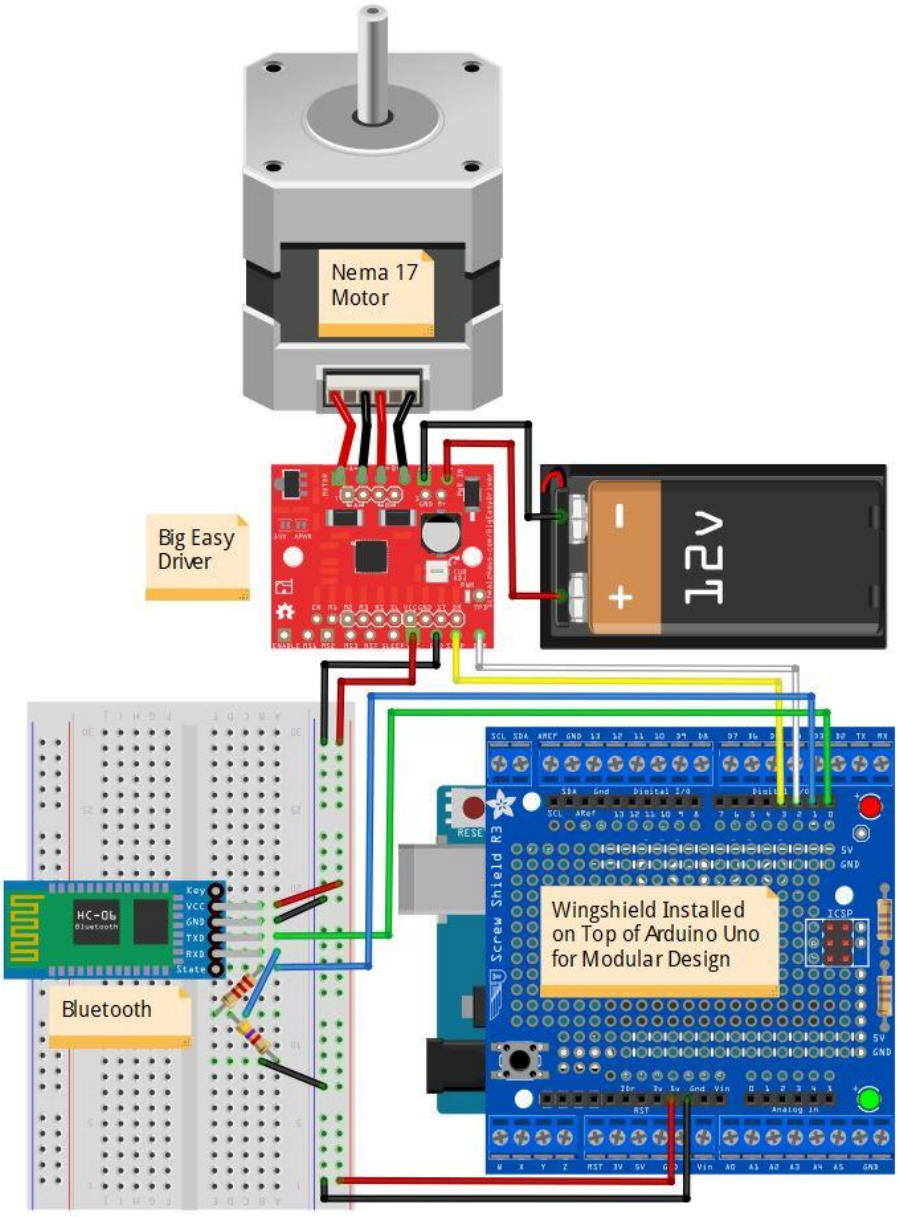

Fig. C.8—Fritzing diagram of syringe pump electrical components.

1) Arduino Uno R3

The Arduino Uno is a microcontroller board based on the ATmega328P. A circuit schematic of the Arduino Uno is shown in **Fig. C.9**. It has a 16 MHz quartz crystal, 6 analog inputs, 14 digital input/output pins, a USB connection port, a power jack, an ICSP header and a reset button. Power is supplied through either the USB connection port or the power jack. Since the system must be isolated, a separate 9V power supply is used to power the Arduino Uno during scanning. The pins operate at 5V with each pin capable of providing and receiving a maximum of 40 mA. Of all the specialized pins, a few are particularly relevant to this project. The serial pins of 0 (RX) and 1 (TX) receive and transmit TTL serial data, which are used to communicate with the Bluetooth module.

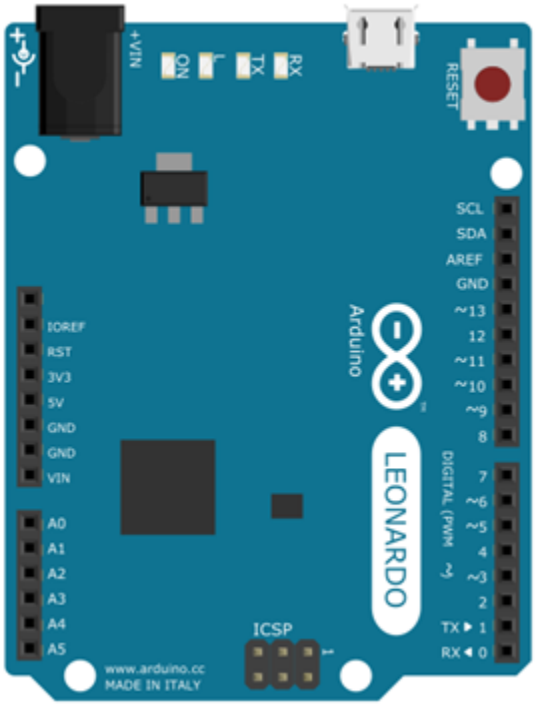

Fig. C.9—Fritzing breadboard schematic of Arduino Uno.

3) HC-06 Bluetooth Module

A Bluetooth module is needed for the Android phone or computer to communicate with the Arduino Uno. An app is used to send data from an Android phone or computer to the Bluetooth module, which is connected to the Arduino Uno via a serial connection. MIT App Inventor 2 is used to create an app to control the Arduino, as detailed in Section C.4. Tera Term is used for communication between a computer and the Arduino.

The Bluetooth module has a VCC, GND, TXD, RXD, and reserve LED status output pin. The module has a current of 30 mA unpaired, 10mA paired and a working voltage from 3.3 to 6V, so the VCC pin of the Bluetooth can be connected to the 5V output pin of the Arduino for powering the module. The GND pin of the Bluetooth module can be connected to any of the GND pins on the Arduino. The TXD pin, which is used to send data from the module to the Arduino, needs to be connected to the serial receive pin (RX) of the Arduino. Similarly, the RXD pin, which is used to receive data from the Arduino, needs to be connected to the serial Arduino transmit pin (TX).

For the HC-06 Bluetooth module, the interface level for TXD and RXD is 3.3V, meaning that while the module can still be powered with 5V from the Arduino Uno, the communication lines to and from the module are only rated up to 3.3V. Sending data from the Bluetooth module to the Arduino (TXD to RX) will not be an issue since Arduino's RX line can interpret the 3.3V signal from the Bluetooth. However, when the Arduino is transmitting data to the Bluetooth (TX to RXD), the signal will be at 5V, which is higher than the working interface level of 3.3V. This may damage the module. To prevent possible damage, a voltage divider is used to decrease the voltage of the Arduino TX line from 5V to approximately 3.3V. The voltage divider equation is shown in **Eq. C-4**, with 2.2 kΩ resistor as R_1, and 4.7 kΩ resistor as R_2. A circuit schematic of the voltage divider and Bluetooth module is shown in **Figs. C.10 and C.11**, respectively.

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \ldots\ldots (C\text{-}4)$$

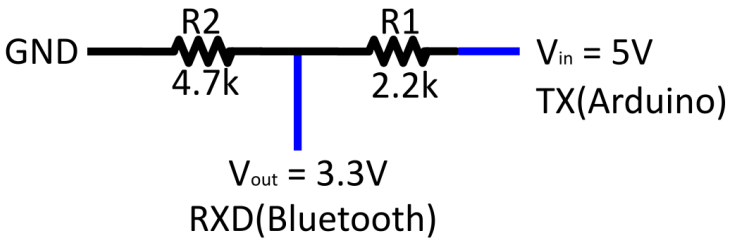

Fig. C.10—Voltage divider schematic.

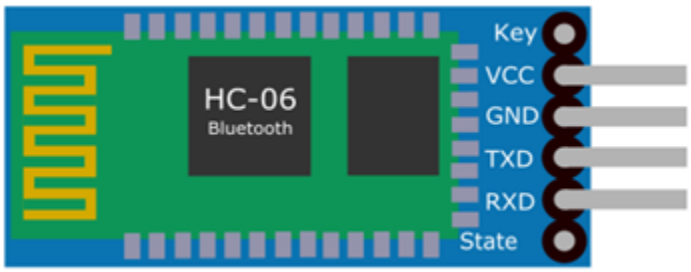

Fig. C.11—Fritzing breadboard schematic of HC-06 bluetooth module.

Arduino source code is written to specify how the Android phone or computer will communicate with the Arduino. Refer to Section C.3 for details on coding. Once the code is written, the code can be uploaded to the Arduino through the on-board port. This process, however, requires the TX and RX lines to be unplugged since the Arduino uses the serial communication lines during uploading so the pins RX (digital pin 0) and TX (digital pin1) are busy. Once the code is uploaded, the Bluetooth must be paired with the Android phone during first time use. The default password for the HC-06 Bluetooth module is 1234.

3) Big Easy Driver

Similar to the Allegro A4988 stepper driver, this stepper motor driver is designed for bi-polar stepper motors like the NEMA 17 motor up to a maximum of 2A/phase (**Fig. C.12**). The Big Easy Driver defaults to 16 step micro-stepping mode, can take a maximum voltage of 30V, and

includes an on-board 5V/3.3V regulation. The on-board regulation allows a single power supply to power both the driver and Arduino microcontroller. In this project, a 12V battery is used. A heatsink is recommended for operations at loads approaching 2A/phase. In operation, a fan is also helpful in dissipating heat. The stepper motor's four wires should be connected to the Big Easy Driver, power supply routed to the power pins, and connections made between the STEP and DIR pin of the driver to the digital pins of the Arduino.

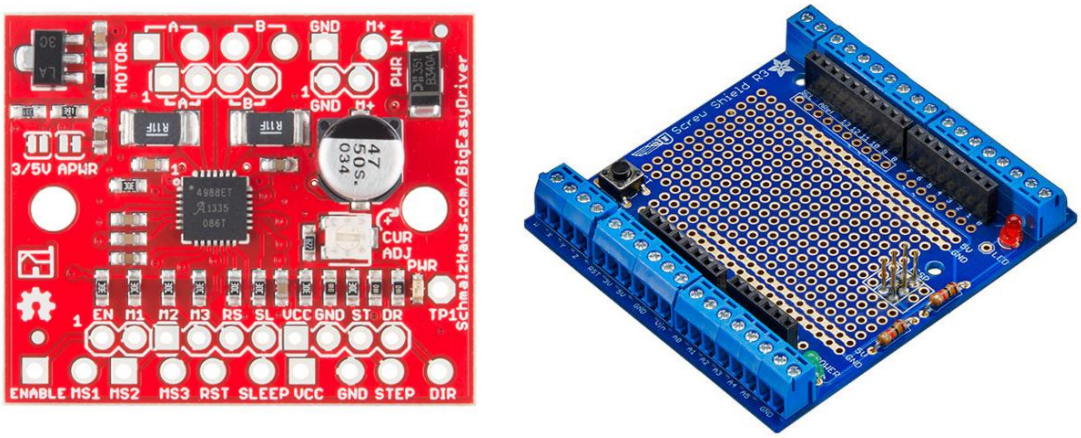

Fig. C.12—Picture of Big Easy Driver (left) and proto-screwshield (right).

4) Proto-Screwshield (Wingshield)

Stacking the prototyping shield (**Fig. C-12**) over the Arduino gives a more modular design and easy access to the existing analog and digital ports of the Arduino. Also, the Big Easy Driver fits squarely on top of the shield. While this part is not necessary in order for the syringe pump to function correctly (unlike the aforementioned parts), efficient use of space is maximized by stacking these parts together.

C.3 Arduino IDE for Syringe Pump

The Arduino Integrated Development Environment (IDE) is a software coding tool that allows users to connect and communicate to the Arduino hardware through uploaded sketches. digitalWrite() and digitalRead() are all examples of functions that are used within the IDE

environment to control devices connected to any of the Arduino input/output pins. For example, digitalWrite(5,HIGH) triggers a HIGH reading to whatever is connected to pin 5.

The sketch for the syringe pump, which is shown in Appendix A.2, is divided into three sections. The first section specifies all relevant variables and libraries. This includes variables like 'Received' for storing any incoming data from the serial port via Bluetooth connection, variables like 'MotorDirection' for keeping track of the state of motor rotation, variables like 'Mode' for turning on/off the motor. The digital pins to which the Big Easy Driver are connected to are also defined in this section. For example, the STEP pin is connected to digital pin 3 and the DIR pin is connected to digital pin 2 of the Arduino.

The setup section follows next. This section is called once when the sketch is first read. It is used to initialize variables, pin modes, libraries that were defined in the previous section. In this section, the baud rate or serial communication rate is set to 9600 as required by the Bluetooth module. The pin modes specified in the previous section are defined as outputs and their initial values set (i.e. DIR pin is defined as output and set to HIGH to default into injection and not withdraw mode).

The loop section is the last part of the code. This part of the code loops continuously, allowing it to actively control the Arduino. For control of the syringe pump, incoming data, when sent from the Android phone or computer via Bluetooth, is stored by the Arduino Uno as the 'Received' variable. If the character '1 cc/min' is received, which is sent from the Android app when the 'Rate_1cc_min' button is pressed or from the computer via Tera Term, Arduino will set the delay to 11.18 ms. Refer to Section C.4 for details on the design of the Android app. Similarly, if the character 'Withdraw' is received, which is sent from the Android app when the 'Withdraw' button is pressed, Arduino will set the DIR to LOW thereby reversing the motor direction.

C.4 Android App for Bluetooth Communication

MIT App Inventor 2 is used to design a compatible app for the aforementioned Arduino sketch. **Fig. C.13** shows a screenshot of the finished application.

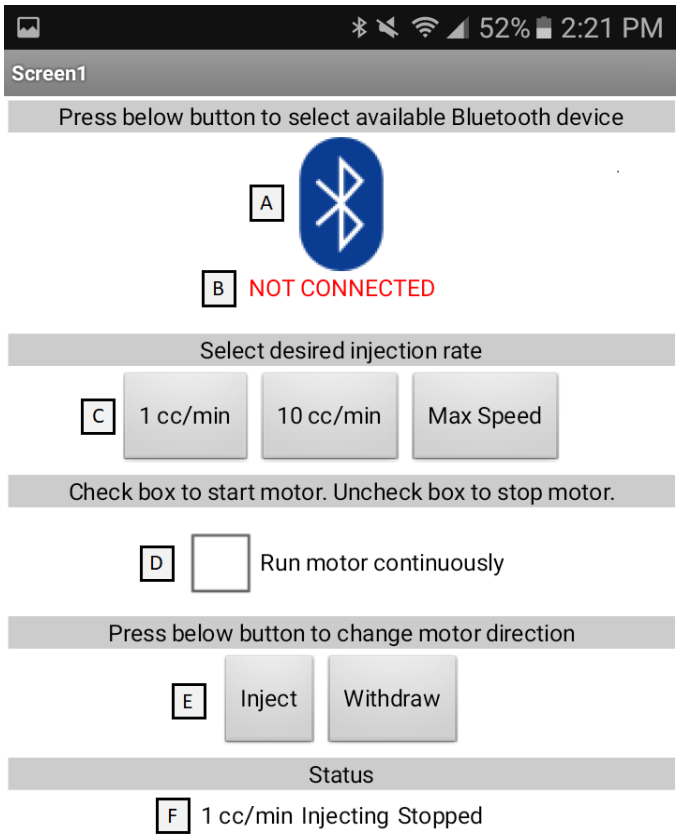

Fig. C.13—Screenshot of 'SyringePump' Android application with markers A-F.

The first step to creating the application is to design the graphical user interface using the 'Designer' tab. 'HorizontalArrangements' from the 'Layout' palette is chosen and its properties (height, width, alignment) are manipulated to match the desired look. Then, buttons and labels are added to the desired 'HorizontalArrangements'. Buttons and labels are selected from the 'User Interface' Palette.

For the Bluetooth interface, from the UserInterface Palette, a ‘ListPicker’ is added and named ‘Bluetooth_List’. A Bluetooth icon is attached as an image as shown with the marker ‘A’. The ListPicker is used to select nearby Bluetooth devices to connect to. The marker ‘B’ is a label named ‘Connected_Label’. This label indicates the connection status – whether the Bluetooth is ‘CONNECTED’ or ‘NOT CONNECTED’ to the phone. Two non-visible components are also added: the ‘BluetoothClient’ from the ‘Connectivity’ palette as well as a ‘Clock’ from the ‘Sensors’ palette, which will be used for the real time indication of the connection status.

Using the ‘Blocks Editor’ tab, blocks or function commands are assigned to the previously added components. **Fig. C.14** shows the two blocks corresponding to the ‘Bluetooth_List’ or marker ‘A’. The top block prompts a list of paired Bluetooth devices. The bottom block allow the phone to connect to a previously selected Bluetooth address.

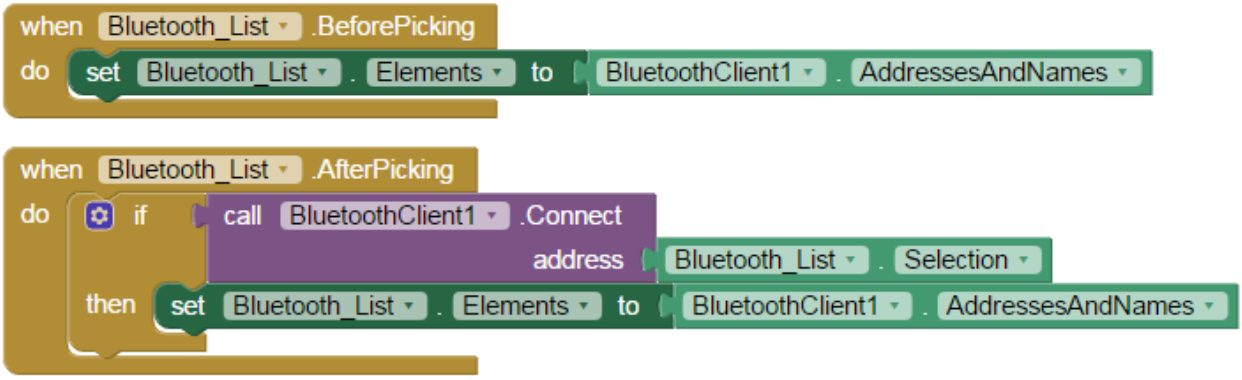

Fig. C.14—Logic blocks corresponding to ‘Bluetooth_List’ or marker ‘A’.

Fig. C.15 shows the block corresponding to the ‘Clock’ and ‘Connected_Label’ or marker ‘B’. This block allows for real time indication of the connection status of the Bluetooth. **Fig. C.16** shows the Bluetooth is connected to the Android phone.

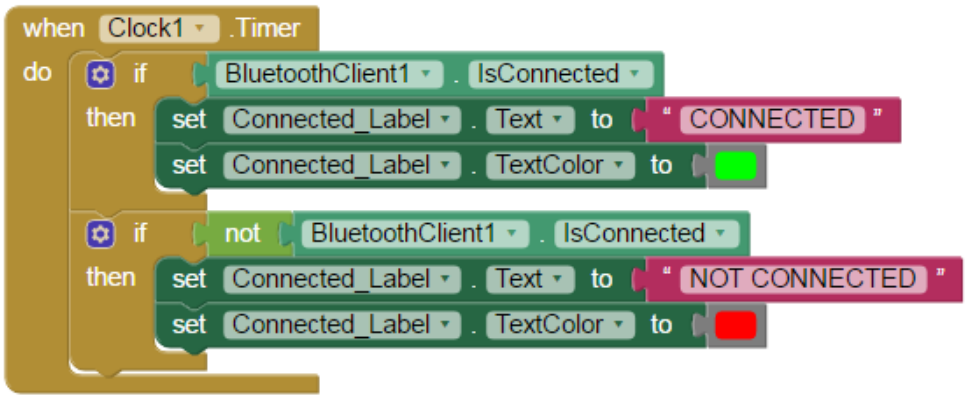

Fig. C.15—Logic blocks corresponding to 'Clock' and 'Connected_Label' or marker 'B'.

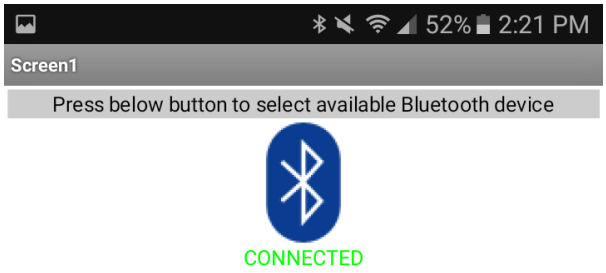

Fig. C.16—Screenshot of 'WAG' Android application indicating the Bluetooth is activated.

The remaining markers C-F correspond to buttons and labels that send data to the Bluetooth module and Arduino Uno to control the injection rate, motor mode, and motor direction. Like mentioned in Section C.3, incoming data, which is sent from the Android phone via the serial port, is stored by the Bluetooth module and Arduino Uno as the 'Received' variable. The incoming data are string values. For this app, the string '1 cc/min' can only be sent when 'Rate_1cc_min' button is pressed. This button corresponds to marker 'C' of Fig. C-11. Once this button is triggered, 'InjectionRate_Label' or marker 'F' is changed to "1 cc/min". The corresponding code block to execute this is shown in **Fig. C.17**.

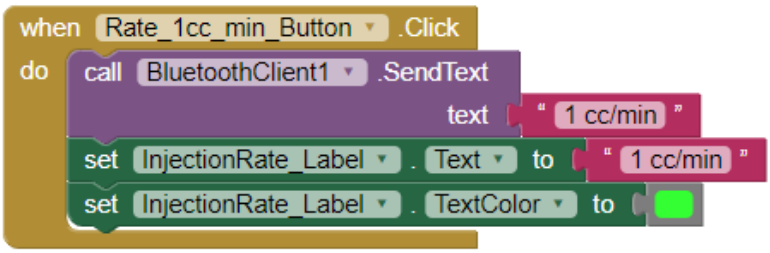

Fig. C.17—Logic blocks corresponding to marker C.

A check box and label is also created for the motor mode – whether the motor is running continuously or not. The string 'Start' can only be sent when the check box is checked. This button corresponds to marker 'D' of Fig. C-11. Once triggered, 'Mode_Label' or marker 'F' is changed to "Running". The corresponding code block is shown in **Fig. C.18**.

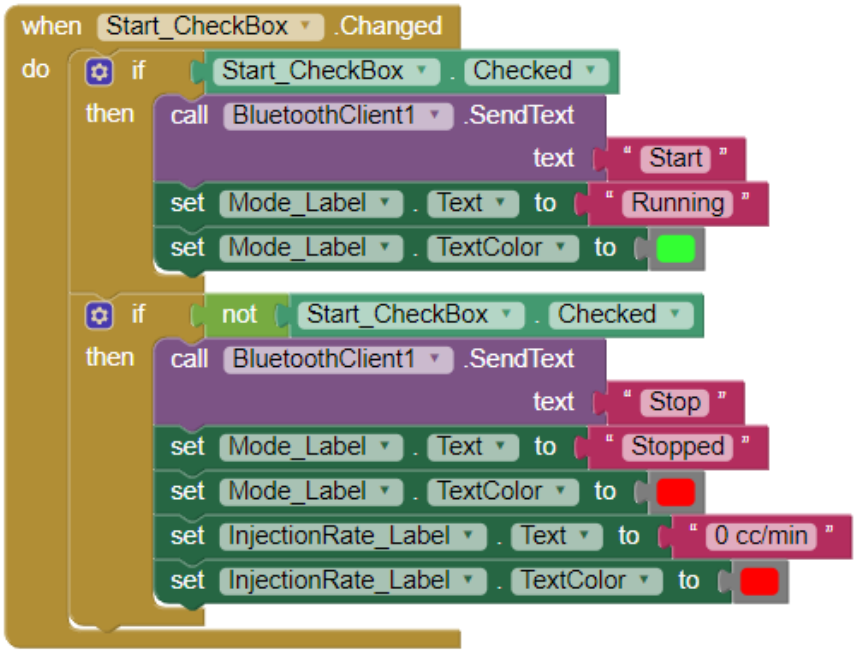

Fig. C.18—Logic blocks corresponding to marker D.

Buttons and labels are also created for changing motor direction. The string 'Inject' can only be sent when 'Inject_Button' is pressed. This button corresponds to marker 'E' of Fig. C-11. Once triggered, 'MotorDirection_Label' or marker 'F' is changed to "Injecting". The corresponding code block is shown in **Fig. C.19**.

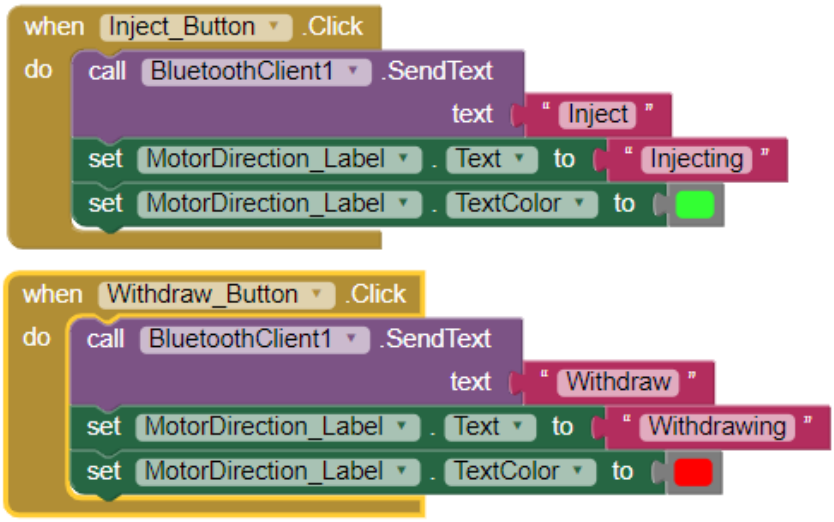

Fig. C.19—Logic blocks corresponding to markers E.

References

Bodziak, R., Clemons, K., Stephens, A., and Meek, R. 2014. The Role of Seismic Attributes in Understanding the Hydraulically Fracturable Limits and Reservoir Performance in Shale Reservoirs: An Example from the Eagle Ford Shale, South Texas: AAPG Bulletin 98(11): 2217–2235.

Chan, G. 2018. Using MDD10A with Arduino Uno, https://tutorial.cytron.io/2015/04/05/using-mdd10a-with-arduino-uno/ (accessed 4/15/2019)

Chen, X., Verma, V., Espinoza, D.N., and Prodanovic, M. 2017. Pore-Scale Determination of Gas Relative Permeability in Hydrate-Bearing Sediments Using X-Ray Computed Micro-Tomography and Lattice Boltzmann Method. Water Resources Research 54: 600–608.

Chen, X., Espinoza, D.N., Tisato, N., and Flemings, P.B. 2018a. Pore-Scale Evidence of Ion Exclusion and Brine Salinity Change During Methane Hydrate Growth in Sandy Sediments. Manuscript submitted for publication.

Chen, X., and Espinoza, D.N. 2018b. Ostwald Ripening Changes the Pore Habit and Spatial Variability of Clathrate Hydrate. Fuel 214: 614-622.

Chen, X., and Espinoza, D.N. 2018c. Surface Area Controls Gas Hydrate Dissociation Kinetics in Porous Media. Fuel 234: 358-363.

Christensen, J. R., Stenby, E. H., and Skauge, A. 1998. Review of WAG Field Experience. Presented at the International Petroleum Conference and Exhibition of Mexico, Villahermosa, Mexico, 3-5 March. SPE-39883-MS.

Christie, M. A., Muggeridge, A. H., and Barley, J. J. 1993. 3D Simulation of Viscous Fingering and WAG Schemes. SPE Res Eng 8(1): 19–26. SPE-21238-PA.

Dai, S., and Seol, Y. 2014. Water Permeability in Hydrate-Bearing Sediments: A Pore-Scale Study. Geophysical Research Letters 41: 4176–4184.

Demin, W., Jiecheng, C., and Qingyan, Y. et al. 2000. Viscous-Elastic Polymer Can Increase Microscale Displacement Efficiency in Cores. Presented at the SPE Annual Technical Conference and Exhibition, Dallas, Texas, 1-4 October 2000. SPE-63227-MS.

Duan, Z. H., Moller, N., Greenberg, J., and Weare, J. H. 1992. The Prediction of Methane Solubility in Natural-Waters to High Ionic-Strength from 0°C to 250°C and from 0 to 1600 Bar. Geochimica et Cosmochimica Acta 56(4): 1451-1460

Duan, Z. H., and Mao, S. D. 2006. A Thermodynamic Model for Calculating Methane Solubility, Density and Gas Phase Composition of Methane-Bearing Aqueous Fluids from 273 to 523 K and from 1 to 2000 bar, Geochimica et Cosmochimica Acta, 70(13), 3369-3386

EIA. U.S. Energy Information Administration. 2018. Annual Energy Outlook.

Emrani, A. S., Ibrahim, A. F., and Nasr-El-Din, H. A. 2017. Mobility Control using Nanoparticle-Stabilized CO2 Foam as a Hydraulic Fracturing Fluid. Presented at the 79th EAGE Conference and Exhibition, Paris, France, 12-15 June. SPE-185863-MS.

Gale, J.F.W., Reed, R.M., and Holder, J. 2007. Natural Fractures in the Barnett Shale and Their Importance for Hydraulic Fracture Treatments: AAPG Bulletin, 91, 603-622.

Germanovich, L.N., and Astakhov, D.K. 2004. Fracture Closure in Extension and Mechanical Interaction of Parallel Joints. Journal of Geophysical Research 109(B2).

Gerward L. X-ray Attenuation Coefficients: Current State of Knowledge and Availability. 1993. Radiat Phys Chem 41: 783–789

Gogarty, W. B. 1967. Mobility Control with Polymer Solutions. SPE J. 7(2): 161–173. SPE-1566-B.

Jennings, R. R., Rogers, J. H., and West, T. J. 1971. Factors Influencing Mobility Control by Polymer Solutions. J Pet Technol 23(3): 391–401. SPE-2867-PA.

Jones. 1972. A Rapid Accurate Unsteady State Klinkenberg Permeameter. SPE J. 12(5): 383-397. SPE-3535-PA.

Kang, D. H., Yun, T. S., Kim, K. Y., and Jang, J. 2016. Effect of Hydrate Nucleation Mechanisms and Capillarity on Permeability Reduction in Granular Media. Geophysical Research Letters 43: 9018–9025.

Karlsen, H. 2010. Library: DS3231, http://www.rinkydinkelectronics.com/library.php?id=73 (accessed 8/01/2017)

Ketcham, R.A. and Carlson, W.D. 2001. Acquisition, optimization and interpretation of X-ray CT imagery applications to the geosciences. Computers and Geosciences 27: 381-400.

Ketcham, R.A., Slottke, D.T., and Sharp, J.M. 2010. Three-dimensional measurement of fractures in heterogeneous materials using high-resolution X-ray computed tomography. Geosphere 6(5): 499-514.

Kim, I., Worthen, A. J., Lotfollahi, M. et al. 2016. Nanoparticle-Stabilized Emulsions for Improved Mobility Control for Adverse-mobility Waterflooding. Presented at the SPE Improved Oil Recovery Conference, Tulsa, Oklahoma, 11-13 April. SPE-179644-MS.

Klauda, J. B., and Sandler, S. I. 2005. Global Distribution of Methane Hydrate in Ocean Sediment. Energy and Fuels 19(2): 459–470.

Kneafsey, T. J., Seol, Y., Gupta, A., and Tomutsa, L. 2011. Permeability of Laboratory-Formed Methane-Hydrate-Bearing Sand: Measurements and Observations using X-ray Computed Tomography. SPE Journal 16(01): 78–94.

Kovscek, A. R., Tadeusz, W. P., and Radke, C. J. 1997. Mechanistic Foam Flow Simulation in Heterogeneous and Multidimensional Porous Media. SPE J. 2(4): 511–526. SPE-39102-PA.

Kovscek, A.R. and Radke, C.J. 1994. Fundamentals of Foam Transport in Porous Media. Foams: Fundamentals and Applications in the Petroleum Industry, L.L. Schramm ed. 115-163. Washington, DC: Advances in Chemistry Series 242, American Chemical Soc.

Krause, R.E., Lane, R.H., Kuehne, D.L. et al. 1992. Foam Treatment of Producing Wells To Increase Oil Production at Prudhoe Bay. Presented at the SPE/DOE Enhanced Oil Recovery Symposium, Tulsa, Oklahoma, 22-24 April 1992. SPE-24191-MS.

Krevor, S., Blunt, M. J., Benson, S. M., et al. 2015. Capillary Trapping for Geologic Carbon Dioxide Storage – From Pore Scale Physics to Field Scale Implications. International Journal of Greenhouse Gas Control. 40: 221–237.

Kumar, A., Maini, B., Bishnoi, P. R., Clarke, M., Zatsepina, O., and Srinivasan, S. 2010. Experimental Determination of Permeability in the Presence of Hydrates and its Effect on the Dissociation Characteristics of Gas Hydrates in Porous Media. Journal of Petroleum Science and Engineering 70(1–2): 114–122.

Kvenvolden, K. A. 1988. Methane Hydrate—A Major Reservoir of Carbon in the Shallow Geosphere? Chemical Geology 71(1): 41–51.

Lee, H.P., Olson, J.E., Holder, J., Gale, J.F.W., Myers, R. 2015. The Interaction of Propagating Opening Mode Fractures with Pre-Existing Discontinuities in Shale. J. Geophys. Res. 120(1): 169–181.

Martinsen, H. A., and Vassenden, F. 1999. Foam for Gas Mobility Control in the Snorre Field: The FAWAG Project. SPE-56478-MS.

Mokhtari, M., Bui, B.T., and Tutuncu, A.N. 2014. Tensile Failure in Shales: Impacts of Layering and Natural Fractures. Presented at the SPE Western North American and Rocky Mountain Joint Meeting, Denver, Colorado, 17-18 April. SPE-169520.

Naroom. 2014. Open Syringe Pump, 2014, https://hackaday.io/project/1838-open-syringe-pump (accessed 25 July 2017).

Nedelkovski, D. 2017. Arduino Tutorials, http://howtomechatronics.com/category/tutorials/arduino (accessed 5 June 2017).

Nguyen, Q. P., Rossen, W. R., Zitha, P. L. J., and Currie, P. K. 2009. Determination of Gas Trapping With Foam Using X-Ray CT and Effluent Analysis. SPE J. 14(2): 222-236. SPE-94764-PA.

Priest, J., Rees, E., and Clayton, C. 2009. Influence of Gas Hydrate Morphology on the Seismic Velocities of Sands. J. Geophys. Res. 114(1). B11205.

Prodanovic, M., Esteva, M., Hanlon, M., Nanda G., and Agarwal, P. 2015. Digital Rocks Portal: A Repository for Porous Media Images.

Sharma, S., Dwivedi, V. K., and Pandit, S. N. 2014. A Review of Thermoelectric Devices for Cooling Applications. International Journal of Green Energy. 11(9): 899-909.

Simjoo, M., Dong, Y., Andrianov, A. et al. 2013. Novel Insight Into Foam Mobility Control. SPE J. 18(3): 416-427. SPE-163092-PA.

Sloan, E. D. 1998. Clathrate Hydrates of Natural Gases. Marcel Dekker, New York

Specht, E. 2018. Packings of Equal and Unequal Circles with Maximum Packing Density, http://www.packomania.com (accessed 3 March 2019)

Suarez-Rivera, R., Burghardt, J., Stanchits, S., Edelman, E., and Surdi, A. 2013. Understanding The Effect of Rock Fabric on Fracture Complexity for Improving Completion Design and Well Performance. Presented at the International Petroleum Technology Conference, Beijing, China, 26-28 March.

Sun Z., Jang J., and Santamarina, J.C. 2018. Time-Dependent Pore Filling. Water Resoure Res.

Sydansk, R. D. 1994. Polymer-Enhanced Foams Part 1: Laboratory Development and Evaluation. SPE Advanced Technology Series 2(2): 150–159. SPE-25168-PA.

Tishchenko, P., Hensen, C., Wallmann, K., and Wong, C. S. 2005. Calculation of the Stability and Solubility of Methane Hydrate in Seawater. Chemical Geology. 219: 37-52

Tsau, J. S., and Heller, J. P. 1992. Evaluation of Surfactants for CO2-Foam Mobility Control. Presented at the Permian Basin Oil and Gas Recovery Conference, Midland, Texas, 18-20 March. SPE-24013-MS.

Ramos, M.J. 2018. Quantification of Static and Dynamic Mechanical Anisotropy in Fractured and Layered Rock Systems: Experimental Measurement and Numerical Modeling. Ph.D. Dissertation, The University of Texas at Austin, Austin, Texas.

Vandersteen, K., Busselen, B., Van den Abeele, K., and Carmeliet, J. 2003. Quantitative Characterization of Fracture Apertures using Microfocus Computed Tomography.

Applications of X-ray Computed Tomography in the Geosciences: Geological Society of London Special Publication 215: 61–68.
Van Geet, M., and Swennen, R. 2001. Quantitative 3D-Fracture Analysis by Means of Microfocus X-Ray Computer Tomography: An Example from Coal: Geophysical Research Letters 28: 3333–3336.
Wang, H., Rabiei, M., Wang, S., and Cui, G. 2018. Fracture Quantification Method with 3D X-Ray Image - Entropy-Assisted Indicator Kriging Method. Presented at the SPE Western Regional Meeting, Garden Grove, California, 22-26 April. SPE-190045-MS.